

15 July 2023

ATTN: Bobby Lineberry (DoDAAC: W90BWX)

US ARMY Combat Capabilities Command, DEVCOM
Weapons Development and Integrations Directorate
Aviation and Missile Center/FCDD-AMT-I-ET
E-mail: bobby.i.lineberry.civ@army.mil

Subject: Contract HR0011-21-C-0236 TALONS Program (SBIR Phase II) CDRL A001 Final
Report and Software Deliverables

Dear Mr. Lineberry:

Enclosed is the Final Report for the subject contract. Also included are the software/report deliverables described in the below table. The location of the deliverables are identified in the table, and instructions on how to use the code can be found in the applicable section of the final report as well as the readme files for each of the files. This is the final submittal for this project. PMT considers all deliverables met as required by the subject contract.

Should you have any questions regarding this report or the supporting code, please contact Bryan Grinnell by phone at (734) 277-8410, or by email at bgrinnell@pacmartech.com.

Deliverable	Type	SoW Task	Location
T-SAL formal specification	Report	7	Section 4 of Final Report
T-SAL interpreter	Software	7	File: tsal-main.zip
Formal Definitions of Novelty Levels	Report	9	Final Report Section 4.3
T-SAL R-transformation operators	Software	8	File: noveltygen-main.zip
T-transformation procedures	Software	8	File: noveltygen-main.zip
Evaluation procedures	Software	8	File: test.py and novelty_generator.py
Procedures for collecting metrics	Software	8	File: novelty_generator.py
Procedures for measuring performance	Software	8	File: novelty_generator.py
Procedures for carrying out empirical protocol	Software	8	File: test.py and novelty_generator.py
Draft final project report	Report	10	Sent on 6/30/23
Final project report	Report	10	Enclosed
Technical progress reports	Report	10	Sent Monthly
Financial progress reports	Report	10	Sent Monthly

Sincerely,

Bryan Grinnell

Bryan Grinnell
Program Manager

Title	Attn:	E-mail / :
DARPA/DSO	Dr. Ted Senator	ted.senator@darpa.mil
DARPA/DSO	ADPM	DSO_ADPM@darpa.mil
Contracting Officer	Catherine Stevens	Catherine.stevens@darpa.mil
Contracting Officer's Representative (COR)	Bobby Lineberry (DoDAAC: W90BWX)	bobby.i.lineberry.civ@army.mil
DARPA Portal	DARPA Portal	https://sbir.darpa.mil

Copy (letter only)

Title	Attn:	E-mail:
Administrative Contracting Officer (ACO)	DCMA Carson	jaime.guzman18.civ@mail.mil

Enclosure:

HR001121C0236FINALREPORT- Final report

Attached Files:

tsal-main.zip
noveltygen-main.zip
artue-discoverhistor-ACS2023Experiment
novelty-generator-ASC2023Experiment
tsal-interpreter-ACS2023Experiment

PacMar Technologies

841 Bishop St., Suite 1110

Honolulu, Hawaii 96813

ISRN MDG/TR--2023/10107/2

Technical Report

14 July 2023

Testing AI Learning in Open-world Novelty Scenarios (TALONS) SBIR Phase II Final Report

Program Manager:
Bryan Grinnell

Principal Investigator:
Mark Bercasio

Contract No. HR001121C0236

Prepared for:
Defense Advanced Research Projects Agency (DARPA)

“The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.”

SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies F.K.A Martin Defense Group

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

This page intentionally left blank.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering, and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 07/14/23		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 07/16/2022 - 07/15/23		
4. TITLE AND SUBTITLE Testing AI Learning in Open-world Novelty Scenarios (TALONS) SBIR Phase II Final Report				5a. CONTRACT NUMBER HR001121C0236		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Bryan Grinnell (PMT), Mark Bercasio (PMT), Allison Wong (PMT), Dustin Dannenhauer (Parallax), and Matthew Molineaux (Parallax)				5d. PROJECT NUMBER		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) PacMar Technologies 841 Bishop St., Suite 1110 Honolulu, HI 96813				8. PERFORMING ORGANIZATION REPORT NUMBER ISRN MDG/TR-42023/10107/2		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA 675 N. Randolph Street Arlington VA 22203-2114				10. SPONSOR/MONITOR'S ACRONYM(S) DARPA		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT The work performed under this contract is considered to be fundamental research, and it is, therefore, not subject to publication restrictions. Papers resulting from unclassified contracted fundamental research are exempt from prepublication controls and requirements, pursuant to DoD Instruction 5230.27 dated October 6, 1987.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT This report transmits the results of the Option period of the Phase II continuance of the TALONS SBIR. PacMar Technologies (PMT) and Parallax Advanced Research extended the development of a set of domain independent theories, principled techniques, and software tools for generating, characterizing, and evaluating novelty. These theories and techniques covering novelties are domain independent. The agent evaluation software tool developed under this SBIR is described in this report as well. The framework developed to support the self-driving car domain (CARLA) in the base period was extended to be compatible with the Monopoly domain, Vizdoom domain, and a PMT maritime domain. Efforts under this report were to support the DARPA SAIL-ON program.						
15. SUBJECT TERMS SAIL-ON, Novelty, V&V, CARLA, AI, T-SAL, TSAL, Parallax, PacMar, PacMar Technologies, PMT, TALONS						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Bryan Grinnell	
U	U	U	UU	310	19b. TELEPHONE NUMBER (include area code) 734-277-8410	

Final Report

Testing AI Learning in Open-world Novelty Scenarios (TALONS)

Contents

1. EXECUTIVE SUMMARY	6
1.1 PROJECT SUMMARY	6
2. TASK OBJECTIVES	9
3. TECHNICAL PROBLEM AND LITERATURE REVIEW	11
3.1 TECHNICAL PROBLEM	11
3.2 LITERATURE REVIEW	11
4. GENERAL METHODOLOGY	13
4.1 NOVELTY GENERATION	13
4.1.1 <i>Novelty Definitions</i>	14
4.1.2 <i>Transformations</i>	15
4.2 FORMALISM	15
4.2.1 T-SAL DOMAINS	16
4.2.2 <i>T-SAL States, Goals, and Scenario Generators</i>	23
4.2.3 <i>T-SAL Transformations and Transformation Sequences</i>	26
4.2.4 <i>T-SAL R-Transformations</i>	26
4.2.5 <i>T-SAL T-Transformations</i>	27
4.2.5.1 T--Transformations Examples	27
4.2.5.2 T--Transformations Examples	28
4.2.6 <i>Object Generator Draw Functions</i>	29
4.2.7 <i>Value Generators</i>	29
4.2.8 <i>Fluent Generators</i>	30
4.3 APPLICATION TO NOVELTY HIERARCHY	32
4.3.1 OBJECTS	33
4.3.2 AGENTS	35
4.3.3 ACTIONS	36
4.3.4 RELATIONS	37
4.3.5 INTERACTIONS	38
4.3.6 ENVIRONMENT	39
4.3.7 GOALS	41
4.3.8 EVENTS	41
5. PUBLICATIONS DEVELOPED	42
5.1 AN ENVIRONMENT TRANSFORMATION-BASED FRAMEWORK FOR COMPARISON OF OPEN-WORLD LEARNING AGENTS	42
5.2 TRANSFORMING ENVIRONMENTS TO EVALUATE AGENT ADAPTATION	43
5.3 A FRAMEWORK FOR CHARACTERIZING NOVEL ENVIRONMENT TRANSFORMATIONS IN GENERAL ENVIRONMENTS	43
5.4 HUMAN IN THE LOOP NOVELTY GENERATION	43
6. SAIL-ON TA2 NOVELTY ANALYSIS	44
7. SOFTWARE	45
8. TECHNICAL RESULTS	46
8.1 MONOPOLY DOMAIN	46
8.1.1 AGENT SETUP	47
8.1.2 EVALUATION SETUP	47

PacMar Technologies
TALONS Final Report

8.1.3 NOVELTY LEVEL SETUP	48
8.2 VIZDOOM DOMAIN	49
8.2.1 AGENT SETUP	49
8.2.2 EVALUATION SETUP	50
8.2.3 NOVELTY LEVEL SETUP	50
8.3 MARITIME DOMAIN.....	50
9. IMPORTANT FINDINGS AND CONCLUSIONS.....	51
10. SIGNIFICANT HARDWARE DEVELOPMENT.....	54
11. SPECIAL COMMENTS.....	54
12. IMPLICATIONS FOR FURTHER RESEARCH	54
13. OTHER ITEMS OF INTEREST.....	56
13.1 BUDGET BREAKDOWN	56
14. REFERENCES.....	57
15. APPENDICES	60
APPENDIX A: PARALLAX PUBLICATION: “AN ENVIRONMENT TRANSFORMATION-BASED FRAMEWORK FOR COMPARISON OF OPEN-WORLD LEARNING AGENTS”	60
APPENDIX A: PARALLAX PUBLICATION: “AN ENVIRONMENT TRANSFORMATION-BASED FRAMEWORK FOR COMPARISON OF OPEN-WORLD LEARNING AGENTS”	60
APPENDIX B: PARALLAX PUBLICATION: “TRANSFORMING ENVIRONMENTS TO EVALUATE AGENT ADAPTATION”	60
APPENDIX C: PARALLAX PUBLICATION: “A FRAMEWORK FOR CHARACTERIZING NOVEL ENVIRONMENT TRANSFORMATIONS IN GENERAL ENVIRONMENTS”	60
APPENDIX D: PACMAR TECHNOLOGIES & PARALLAX PUBLICATION “HUMAN IN THE LOOP NOVELTY GENERATION”	60
APPENDIX E: NOVELTY ANALYSIS RESULTS ISI TEAM	60
APPENDIX F: NOVELTY ANALYSIS RESULTS UTD TEAM	60
APPENDIX G: NOVELTY ANALYSIS RESULTS WSU TEAM	60

1. Executive Summary

This project was in support of the DARPA Science of Artificial Intelligence and Learning for Open-world Novelty (SAIL-ON) Program. In the Phase II base period effort, we advanced our theory and implementation of the Phase I TALONS framework and collected initial empirical results of baseline agent performance on novelties produced from our novelty generator. In the Phase II option period effort, we extended our base period work into a more generalized form of novelty generation & implementation.

The key accomplishments during this work are: the completed development of our Novelty Generator, the performance of an analysis of novelties from 3 different SAIL-ON TA1 teams, the development of a novelty generation-implementation process that is automatic and minimizes human bias, the application our novelty generation process to the Monopoly, Vizdoom, Blocksworld domains which supports the case that our novelty generator is domain independent, and support of the SAIL-ON program. The project summary below and follow-on report goes into more detail on these accomplishments.

1.1 Project Summary

The current state-of-the-art in AI can perform robustly in the presence of known knowns and even known unknowns. However, AI is still unable to reliably detect, characterize, and accommodate novelty in the open world. As the research community continues to strive toward achieving advanced autonomy, there is a need for scientifically grounded methods for evaluating the performance of AI agents in open world environments.

PacMar Technologies (PMT) and Parallax Advanced Research have developed a set of domain independent theories, principled techniques, and software tools for generating, characterizing, and evaluating novelty. These theories and techniques cover novelties that are domain-independent. In the base period of our contract we developed a testbed to evaluate agent performance in response to novelty in a self-driving car domain, and we implemented automatically generated novelties from our software tools into the Monopoly domain provided by the USC SAIL-ON team.

Our novelty generator approach uses principled techniques to automatically generate novel scenarios. These scenarios are loaded into simulated environments that interface with a given third-party AI agent to collect data on the agent's performance. It then evaluates the agent's ability to handle various types of novelty under a variety of different circumstances.

We define novelty as a change in an environment. Briefly, change may either be a change in the transition function or a change of the state space. Our approach has the capability to generate novelty in the transition function (which includes actions and events), state space definition, and observation function. The computational methods for precisely generating novelty fall into two types of transformations¹ in a manner consistent with the Creative Systems Framework (Wiggins,

¹ R and T-transformations are terms from the Creative Systems Framework (Wiggins, 2006) which refer to different approaches to generating creative artifacts. An R-transformation is a change to the 'universe' which defines possible transitions and states (example: cars can now fly when traveling at high speeds) and a T-transformation is a change in the process that generates states, such that a new part of the state space will be explored.

2006). Key to our approach is a characterization of novelty in eight dimensions that supports focusing scenario generation on situations likely to challenge an agent's robustness.

The objectives of our approach are as follows:

1. Develop a scientific basis for the characterization of novelty.
2. Develop techniques to generate novel scenarios that can be used to evaluate AI agents.
3. Determine the feasibility of these techniques and their applicability across domains.

This report serves to provide a clear description of our approach, including our methodology for novelty generation, simulation, and evaluation. Application of our approach to CARLA's self-driving car domain, Monopoly, VizDoom, and a maritime domain helped further the development of both theory and testbed software. We conclude this report with reflections and implications for further research.

In the Phase I effort, we developed a framework for generating novel scenarios grounded in a multi-dimensional characterization of novelty. We formally specified this multi-dimensional characterization, metrics for collecting data on AI agents under test, and methods for evaluating agent robustness to different types of novelty. In the novelty generation process using transformations, we defined 24 function signatures and calculated the upper bound complexity of applying them to generate novelty. We showed that by using both types of transformations to vary scenario generation according to our multi-dimensional characterization of novelty, we are able to reduce the space of novel scenarios significantly. To support the precise generation of novelty, we conceptualized the TALONS Simulator Abstraction Language (T-SAL) to describe environments and individual scenarios in those environments. We use these concepts to support the evaluation of third-party AI agents' robustness to novelty.

In the Phase II base period effort, we advanced our theory and implementation of the TALONS framework and collected initial empirical results of baseline agent performance on novelties produced from our novelty generator. We produced three source code repositories described in the base period final report containing the implementations of (1) The TSAL language (2) the novelty generator (implementing R-transformations) and (3) an evaluation framework using the CARLA self-driving vehicle simulator. From a theory perspective, we performed an initial study of analyzing novelties using the novelty dimensions theory developed in the Phase I effort and we formally defined R-transformations and then used those definitions to formally define the novelty hierarchy levels.

Our unique contributions created under this Phase II Option period effort include the following:

- Provide an updated TSAL Interpreter – A python-based library that converts TSAL language documents into a python class instances. Features include reading, writing, and modifying TSAL language documents. During the option period, we added goal representations to the TSAL problem file document parsing component.
- Revised the T-SAL specifications definition.
- Completed development of our Novelty Generator – A python-based library that uses R-transformations and T-transformations to generate new TSAL domain and problem files. Features include:
 - User ability to select which R-transformations to consider, thereby focusing search for generating specific kinds of novelties.

- Initial T-transformations include generating random scenarios and generating scenarios from a seed scenario with optional specifications to prevent certain predicate types from being varied in new scenarios
 - We implemented a domain-independent filtering capability to determine whether a novelty is relevant – the agent will see reduced performance if it does not address the novelty.
- Formal definitions of R-transformations
- Formal definitions of T-transformations
- Updated the TSAL domain files for the Monopoly domain and created TSAL domain files for the Vizdoom domain, Blocksworld domain, and a Maritime domain.
- Conducted evaluations in Monopoly, Vizdoom, and Blocksworld domain in order to refine operation of our novelty generator and show that it can be used on multiple domains.
- Added scenario generator examples for T-transformation for the Blocksworld domain
- Performed an analysis of novelties from SAIL-ON TA1 teams (WSU, UTD, and ISI) using our novelty dimensions.
- We provided rigorously defined logical definitions of the SAIL-ON novelty levels. These definitions can be used to test whether a novelty is of a particular level.
- Formally define three qualities for finding valuable novelties, we refer to these qualities as “learnability conditions” since they capture whether a novelty can be expected to be “learnable” by an agent: relevance, noticeability, controllability.
- We created a human-in-the-loop process where developers can use our novelty generator to create novelties in other simulators, defined outside of TSAL, such as Monopoly and Vizdoom
- Produced 4 publications discussing our work throughout this period of performance
- Supported Month 36 and Month 42 SAIL-On meetings to include presenting our work at both events.

2. Task Objectives

The objectives of phase II of the option period of the project focus on extending concepts from the base period to handle Phase III novelties. The work to complete this objective was broken up into the 4 main tasks that are described below. The deliverable completion dates and locations are identified in Table 1.

Task 7: T-SAL Features to Support External Novelties

Task seven describes the work on the T-SAL language to support Ph. III novelties. Specifically work to extend the TALONS Simulator Abstraction Language (T-SAL) to support language constructs for goal novelties, event novelties, and environment novelties. This includes providing an updated T-SAL interpreter.

Also included in task 7 work is development of formal semantics for T-SAL that describes the meaning of a T-SAL domain file and problem file in terms of a corresponding state machine. An updated T-SAL Specification will be developed.

Task 8: Scenario Generation for External Novelties

Task eight develops / extends the scenario generation process including developing R-Transformation operators, identification of phase III novelties for evaluation, generation of scenario instances from the novelty examples, developing novelty dimension operators, developing t-transformations for scenario generation, developing evaluation metrics and measures, conducting evaluations, and collaboration & support for SAIL-ON.

Task 9: Characterization of External Novelties

Task nine extends the multi-dimensional theory of novelty to include novel environments, novel goals, and novel events. Also included in task nine is the development of formal definitions of novelty levels.

Task 10: Option Period Program Management

Task six describes the programmatic duties to be upheld during the base period.

Table 1 Deliverable Overview

Deliverable	Type	SoW Task	ECD	ACD	Location
T-SAL formal specification	Report	7	10/14/22	10/14/22	Section 4 of Final Report
T-SAL interpreter	Software	7	7/15/23	10/14/22	File: tsal-main.zip**
Formal Definitions of Novelty Levels	Report	9	10/15/23	10/14/22	Final Report Section 4.3
T-SAL R-transformation operators	Software	8	7/15/23	10/28/22	File: noveltygen-main.zip**
T-transformation procedures	Software	8	7/15/23	4/28/23	File: noveltygen-main.zip**
Evaluation procedures	Software	8	7/15/23	1/28/23	File: test.py** and novelty_generator.py**
Procedures for collecting metrics	Software	8	7/15/23	1/28/23	File: novelty_generator.py**
Procedures for measuring performance	Software	8	7/15/23	1/28/23	File: novelty_generator.py**
Procedures for carrying out empirical protocol	Software	8	7/15/23	1/28/23	File: test.py** and novelty_generator.py**
Draft final project report	Report	10	6/15/23	6/30/23	
Final project report	Report	10	7/15/23	7/15/23*	
Technical progress reports	Report	10	Monthly	Monthly	
Financial progress reports	Report	10	Monthly	Monthly	

*Final report ACD is projected to be on the last day of the POP

**Software Delivery will be with the final version of the Final Report

3. Technical Problem and Literature Review

Artificial intelligence (AI) systems are quickly becoming ubiquitous in both civilian and military products. However, existing AI systems in use in the real-world are still unable to contend successfully with novel situations. Multiple ongoing research programs are attempting to resolve this problem by developing AI systems with the ability to respond appropriately, safely, and robustly to novel situations. However, these programs lack scientifically grounded methods to test these applications and more generally, to measure their robustness.

3.1 Technical Problem

We define novelty as new, potentially unknown circumstances and dynamics relevant to an agent's operation. A basic premise of our research is that encountering novelty is inevitable for all AI systems deployed in the real world because of limitations during their design in modelling the environment, other agent motives and behaviors, and unexpected changes to the world itself. However, current verification and validation (V&V) methods lack a scientific basis for characterizing novelty, which limits their ability to generate or synthesize novel conditions for testing. Current V&V methods also lack adequate metrics for measuring performance in novel situations.

The challenge of verifying and validating AI systems deployed in real world settings is fundamentally about measuring and then predicting how these systems will react in novel situations. Measurement of AI system's robustness to novelty requires the ability to generate novel situations. The TALONS System conceptualized in the Phase I effort provides the theoretical foundation to build a novel scenario generation and evaluation system capable of measuring AI agents' robustness to novelty in high-fidelity simulated environments. The TALONS System is one of the first systems to generate novelty in a manner consistent with the Creative Systems Framework (Wiggins, 2006).

3.2 Literature Review

In this section, we review prior work that informed the development of our framework. In the literature on creativity, novelty is a well-defined concept. Wiggins' (2006) Creative Systems Framework (CSF), which formally defined and extended the creativity system described by Boden (1990) in her book *The Creative Mind*, defines novelty as a property of a creative output which previously did not exist. CSF was the first formal logic-based creativity framework, describing, at the highest level, a conceptual space of universes and artifacts that naturalistic, creative agents create in a societal context. Our framework is both narrower in scope and provides greater detail, ranging over transformations to environments that consist of agents, actions, events, processes, and tasks, commonly found in AI literature. In Wiggins' CSF, $\mathcal{R}$ denotes a set of universe constraining rules, and $\mathcal{T}$ denotes rules for traversing a space. We maintain consistency with Wiggins' prior definitions of $\mathcal{R}$ and $\mathcal{T}$, dividing our transformations into R-Transformations and T-Transformations respectively.

The term *novelty* in AI literature has taken on at least two perspectives: novelty from the lens of an individual agent and novelty that exists from the lens of an environments' history. Research of the former defines novelty relative to an agent's experiences (e.g., Muhammad et al. (2021), Boulton

et al. (2021), Gamage et al., (2021)), while our framework defines novelty as the latter, consistent with work on creativity (Wiggins', 2006). Frameworks with an agent's perspective of novelty are useful because they can identify how prepared an agent is for new challenges. However, they cannot usefully describe how environment challenges differ in a way that cuts across different agents and different knowledge representations. This limits the ability for agent-based notions of novelty to support evaluations on a variety of different AI approaches. For historical and language reasons, both areas of research currently use the term "novelty", but in different ways; therefore, we emphasize our novelty framework is environment-based.

Environment-based novelty is fundamentally different from agent-based novelty. In the theory of open world novelty by Boulton et al. (2021), novelty occurs when an agent experiences an environment sufficiently different from its prior experiences. A key feature being that novelty occurs at a point in time. In environment-based frameworks like that of Wiggins' CSF (2006), novelty exists independent of any given agent's experiences or knowledge. Additionally, our framework does not define novelty as occurring at a point in time, but rather as encompassing the time-independent difference between two environments. This means that novelty can exist without any changes to the observation or state space. While valuable for considering what experiences could be new to a particular agent, two agents cannot truly be compared on the "same" novelty unless they have identical prior experiences; such comparisons are a key feature of our framework.

A need for a theory of environmental change has been proposed to help explain and measure progress in open-world learning. Langley's (2020) work provides a set of broad requirements for such a theory, motivating a theoretical formalism for environments and transformations on them. Our framework provides both, and so can be considered an example "theory" in this regard that describes environmental changes and provides specific language for characterizing how environment changes vary.

Existing novelty generation capabilities for evaluating AI systems are designed in close alignment with simulated environments. Current benchmark domains (Goel et al. (2021), Xue et al., (2022), Kejriwal & Thomas (2021), Balloch et al., (2022)) test an agent's ability to respond to novelty using carefully constructed scenarios by domain experts. This is current standard practice, and each class of transformations constructed applies only to a single environment. This ad-hoc construction does not permit generalization over domains or formal statements about what characteristics apply to the novel scenarios generated. To our best knowledge, our framework is the first to systematically provide infinite novelties that are not hand curated by domain experts. Game design research has explored problems of domain generation and notions of transformations. The Metagame system (Pell, 1992) and EGGG system (Orwant, 2000) automatically generated chess-like games using a complicated system of rules and constraints designed for chess-like games in particular. Smith and Mateas (2010) decoupled the design of a game space from the exploration of that space to find interesting games. The language used to design game spaces was large and flexible compared to the prior chess-like games, but still incorporated many assumptions, such as the existence of a rectangular play space. Our framework's language for expressing domains, based on PDDL, can model many aspects found in the game's representations of these works; the aim of our framework is to produce novelty first and then meet other criteria later (such as being an interesting game). Therefore we focus on a more fundamental representation without worrying about domain specific knowledge beyond representational notions of objects, actions, events, and goals. To sculpt the generative space of

our framework, users would provide constraints on the choice of R and T-transformations as they see fit – and such a discussion is outside the scope of this paper.

Prior work (Molineaux & Dannenhauer, 2021) defined a set of environment transformation characteristics, or “dimensions”, useful for classifying the difference between pairs of original and modified environments. These definitions are based on describing environments using a refinement of the Partially Observable Markov Decision Process formalism, rather than the relational manipulations described here. The dimension-based formalism is concerned primarily with broad characterizations of the start and endpoints rather than describing a specific pathway between them. As such, it is more difficult to verify, and does not account for human-intuitive descriptions of novelty categories such as those found in the novelty hierarchy. However, the upside there is much less flexibility in the dimension-based formalism to describe the same environments and transformations in different ways; as such, dimension values have much less “wiggle room” than an arbitrary T-SAL categorization.

4. General Methodology

We now describe the TALONS system components and the theoretical work accomplished during the Option period. Specific results from evaluations are detailed in Section 5 – Technical Results.

4.1 Novelty Generation

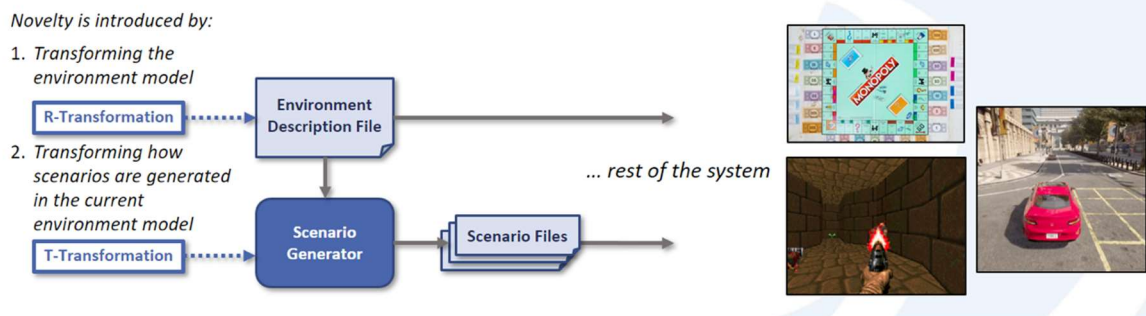


Figure 1: Novelty Generation in the TALONS V&V Framework

To produce novel scenarios to evaluate novelty-aware AI agents, we have devised a first-of-its-kind novelty-based scenario generation process as shown in Figure 1. This process requires initial knowledge in the form of two files describing the default dynamics and starting state of the environment:

1. An initial environment description
2. An initial scenario

The environment description is an abstract, declarative representation of the state space and transition function for a simulated environment, written using the Transformation and Simulator Abstraction Language (T-SAL). T-SAL is inspired by the multiple variants of the Planning Domain Definition Language (PDDL) (Fikes & Nilsson, 1971; McDermott, et al., 1998; Fox & Long, 2003; Edelkamp & Hoffmann, 2004; Gerevini & Long, 2005; Helmert, 2008) for representing states and transitions including actions and events. The scenario is a description of a single instance of the environment. Each scenario has a corresponding environment description

with which it is consistent. We assume that for any simulated environment, there is a provided initial environment description along with an initial scenario². These files serve as the starting point from which we introduce novelty, either through *R-transformations* which are changes to the environment description or *T-transformations* which are changes to the process that generates scenarios (Wiggins, 2006).

The environment description informs both the task generator and the scenario generator to ensure the produced tasks and scenarios are consistent with the current environment description. The scenario generator produces individual scenarios that can be run with the simulator (CARLA self-driving vehicle simulator, Monopoly, Vizdoom). The third-party novelty-aware agent, which we consider to be a black box, runs in the simulation environment instantiated from the produced scenario. During simulation, the agent obtains task and streaming state information from the simulator and sends actions to control the vehicle in the simulator. An individual simulation is run per scenario producing an execution trace. Each execution is then evaluated per metrics. These results could then be accumulated into a final V&V report that would describe the performance of the agent across multiple scenarios.

4.1.1 Novelty Definitions

A primary and unrealized goal of artificial intelligence is *robustness*: the capability to continue to perform well despite novel changes in environments an agent interacts with. Robustness is pivotal to continued autonomy in open worlds – those that are dynamic and unstable like the physical world. To date, the science of machine learning has been assumed to prepare agents to become more robust without rigorously defining what the space of such novel changes are, nor how they can be systematically varied to test the robustness of an agent. Descriptions of concept learning or classification learning problems in machine learning (Mitchell, 1997) traditionally give a set of examples from a fixed distribution, with no described temporal relationships. Reinforcement learning problems (Sutton and Barto, 2018) describe an environment that an agent receives observations from, but generally assume the environment is unchanging. Recent work on open-world novelty (Boult et al, 2021; Muhammad et al, 2021; Gamage et al, 2021), characterizes changes to an environment from the point of view of the agent, preventing direct comparisons across agents with different world views. Langley (2020) describes environmental change, but without a rigorous framework.

To better describe the challenges that a changing environment poses to an agent, we define a general class of environments and *environment transformations* (sometimes referred to as *novelties*). These definitions are useful for 1) describing the extent of environment transformations, 2) categorizing and grouping transformations with similar characteristics, 3) communicating and replicating transformations, and 4) randomly generating novelties that test the robustness of an agent to novel changes in its environment. To our knowledge, this is the first framework of definitions that describes formally and completely a space of environment transformations over a general class of environments.

² Or that it is possible to define these for the chosen agent-environment.

4.1.2 Transformations

Transformations to environments come in two general categories: *R-Transformations* to the dynamics of the environment, and *T-Transformations* to scenario distributions; the latter category includes distributions over possible initial states and performance criteria. To rigorously define a general class of environments and novelties, we start with a well-known and tested formalism for representing these environments, the planning domain definition language (PDDL) family of languages. PDDL provides a general mechanism for describing individual scenarios referred to as “problems”, and a separate mechanism for describing environment dynamics referred to as “domains”. In this work, we formally describe the general notion of a “scenario generator”, which describes a distribution of scenarios rather than a single scenario. Using established PDDL formalisms, we build a new language, the T-SAL (Transformation and Simulator Abstraction Language) Domain language, that covers the broadest class of environments possible.

T-SAL R-Transformations describe changes to domains as used in PDDL. PDDL domains use a subset of first-order logic to describe environment dynamics: changes that may and must happen as a result of agent actions and the natural interactions of objects without agency. Unlike prior work on PDDL languages, we do not constrain the representation of T-SAL based on backwards compatibility or to match well the capabilities of existing automated planners; therefore, we incorporate events and processes from PDDL+ that describe continuous change, non-deterministic effects and events that describe change probabilistically, mathematical operators that describe proportional change, and an object introduction effect that describes open-world change.

T-SAL T-Transformations describe changes to scenario generators. In addition to the initial state classically described within a PDDL problem, these scenario generators describe how to draw initial state objects, and fluents from statistical distributions. Instead of a goal, scenario generators include a performance function. This is similar to existing maximization goals, but defined with respect to agents as well as the state, permitting multiple agents’ performance to be described. T-Transformations and R-Transformations are useful both to generate and to formally describe novelty. This work contributes to the literature on open-world novelty by formally characterizing a group of novelty concepts previously described only informally. Scientists working together in the DARPA SAIL-ON program to organize and schedule program advancement developed a group of characterizations described as a “novelty hierarchy”. This novelty grouping was previously defined only informally, so some practitioners may reasonably disagree with our definitions; however, we believe they capture the essence of the designers’ intent. By providing this formal definition of the novelty hierarchy used in early novelty investigations, we hope to show that T-SAL transformations are a viable means of rigorously categorizing novelty in the future. We believe this forms a suitable basis for future investigations of the impact of novelty categories on agents.

4.2 Formalism

T-SAL is a general term that refers to two languages for describing environments: the computational representation (T-SAL-CR), and the logical language that describes it, L(T-SAL), read as ‘logic of T-SAL’. L(T-SAL) is a first-order logic that describes domains, scenario generators, and environment transformations at an abstract level. It provides common terminology and is used for rigorous definitions. T-SAL-CR describes individual domains, using domain-specific fluents and types to describe the dynamics of domains. We build up both in this

formalism; L(T-SAL) is used to define and make general statements about environments and environment transformations. The T-SAL-CR is provided to communicate examples. An exact semantics for the T-SAL-CR is not provided here; however, it's similar to existing computational representations in the PDDL family (see Appendix C for a detailed comparison). All examples of T-SAL concepts use the well-understood Cart-Pole domain, supplemented with moving blocks and multiple carts.

4.2.1 T-SAL Domains

T-SAL domains describe properties and dynamics of environments. This section builds up the concept of a domain by describing simpler concepts used in the description of a domain in a bottom-up fashion. A BNF for the T-SAL domain language is given in Appendix C. All variables introduced throughout the rest of this section are shown in Table 2. Short definitions and examples of major T-SAL domain concepts are given in Table 3 (T-SAL Specification).

<i>S</i> : the set of symbols	Alt: the set of effect modifications {SET, INCREASE, DECREASE, CREATE}
<i>V</i> : variables in a domain description	
<i>V</i> : the space of variable sets	Dir: the set of change directions {INCREASE, DECREASE}
<i>Te</i> : terms which may be of the form of symbols, variables, integers, or reals,	Ev: the space of events
<i>ty</i> : a type in the space of symbols <i>S</i>	Ev : the space of sets of events
	ev: an event in the space of events <i>Ev</i>
<i>F</i> : the space of functions	<i>P</i> : the space of process models
<i>f</i> : a function in the space of functions <i>F</i>	P : the space of sets of process models
<i>Fn</i> : the set of function names	<i>p</i> : a process model in the space of process models
<i>fn</i> : a function name	<i>D</i> : the space of T-SAL domains
<i>Cn</i> : the space of conditions	<i>d</i> : a domain in the space of T-SAL domains <i>D</i>
Cn : the space of sequences of conditions	<i>st</i> : a T-SAL state that is true in the environment at a particular moment in time
<i>end</i> : a condition in the space of conditions <i>Cn</i>	<i>og</i> : an object generator
<i>Ca</i> : the space of calculations	OG : the space of object generators
Ca : the space of sets of calculations	<i>vg</i> : a value generator
<i>calc</i> : a calculation in the space of calculations <i>Ca</i>	VG : the space of value generators
<i>Ax</i> : the space of axioms	Va : the space of values
Ax : the space of sets of axioms	<i>fg</i> : a fluent generator
<i>ax</i> : an axiom in the space of axioms <i>Ax</i>	<i>sg</i> : a T-SAL scenario generator
<i>E</i> : the space of effects	SG : the space of scenario generators
E : the space of sets of effects	<i>GF</i> : the space of ground fluents
<i>e</i> : an effect in the space of effects <i>E</i>	<i>gf</i> : a ground fluent in the space of ground fluents
<i>C</i> : the space of continuous changes	<i>AG</i> : the space of agents
C : the space of sets of continuous changes	<i>ag</i> : an agent in the space of agents
<i>c</i> : a continuous change in the space of continuous changes <i>C</i>	<i>t</i> : an environment transformation
<i>A</i> : the space of action models	T : the space of transformation sequences
A : the space of sets of actions models	<i>U</i> : the universal set
<i>a</i> : an action model in the space of action models <i>A</i>	

Table 2 – Summary of variables used in L(T-SAL)

Both L(T-SAL) and T-SAL-CR borrow heavily from the PDDL (McDermott, et al. (1998); Fox & Long, (2003); Edelkamp & Hoffmann (2004); Gerevini & Long, (2005); Fox & Long, (2006); Helmert, (2008); PDDL 3.1, PDDL +) family of languages developed in the planning community, which in turn took inspiration from STRIPS (Fikes & Nilsson, 1971) and the situation calculus

(McCarthy and Hayes 1950). While we share the objective of representing environments simply and concretely, our primary objective is to enable a complete description of a range of transformation operators, which necessitates a different set of domain representation decisions. This results in a similar representation to PDDL languages that is not directly descended from any one such language. Ease of planning was not a primary design criterion, and we do not attempt to provide backwards compatibility with existing PDDL languages; instead, T-SAL follows the same guidelines but without historical baggage.

Concept	<i>Var, Space</i>	L(T-SAL) Definition	T-SAL-CR Example
Symbol	S	<enumerated class>	CART
Variable	$v \in V$	$V \subset S$	?C
Term	Te	$Te \equiv V \cup Va \cup \langle Fn, Te \rangle$	?C; CART; 3; 1.5; TRUE; (CART-VELOCITY ?C)
Type	$ty \in Ty$	$Ty \subset S$	CART
Function	$f \in F$	$F \equiv S \times V \times (V \rightarrow S) \times S$	(CART-VELOCITY ?C – CART) – SPEED
Ground Fluent	$gf \in GF$	$GF \equiv Fn \times (V \rightarrow Va) \times Va$ Tuple: $\langle fn, args, value \rangle$	(= (CART-VELOCITY CART1) 0)
Calculation	$calc \in Ca$	$Ca \equiv Te \cup F \times Te \cup Op \times Ca \times$ $Ca \cup AggOp \times V \times Cn \times Ca$ $\cup IF Cn \times Ca \times Ca$ Tuples: $\langle fun, terms \rangle \langle op, calc1, calc2 \rangle$ $\langle SUM, v, con, calc \rangle \langle PRODUCT, v,$ $con, calc \rangle \langle IF, cnd, calc_T, calc_F \rangle$	0; (POLE-ANGLE ?C); (+ (POLE-ANGLE ?C) .01); (/ (SUM ?C ($\neq$ (CART-VELOCITY ?C) 0) (POLE-ANGLE ?C)) (SUM ?C ($\neq$ (CART-VELOCITY ?C) 0) 1))
Condition	$cnd \in Cn$	$Cn \equiv Te \cup PC \cup BOp \times Cn \times Cn$ $\cup FORALL \times V \times Cn \times Cn$ $\cup HAS-VALUE \times V \times Ca$ Tuples: $\langle AND, cnd1, cnd2 \rangle \langle OR, cnd1,$ $cnd2 \rangle$ $\langle FORALL, v, cons, req \rangle \langle HAS-$ $VALUE, v, calc \rangle$	TRUE; (< (POLE-ANGLE ?C) 30); (AND (< (POLE-ANGLE CART1) 30) (> (POLE- ANGLE CART1) -30)); (FORALL ?C ($\neq$ (CART-VELOCITY ?C) 0) (< (POLE-ANGLE ?C) 30)); (HAS-VALUE ?A (POLE-ANGLE ?C))
Effect	$e \in E$	$E \equiv S \times V \times Alt \times Te \times [0..1]$	(DECREASE (POLE-ANGLE ?C) 0.1) [0.5]
Continuous Change	$c \in C$	$C \equiv S \times V \times Dir \times Ca$	(INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C))
Action Model	$a \in A$	$A \equiv S \times Te \times V \times (V \rightarrow Ty) \times Cn$ $\times E$ Tuple: $\langle name, performer, args,$ $argTypes, precs, effs \rangle$	(:ACTION PUSH :PERFORMER ?AG :PARAMETERS (?C – CART) :PRECONDITIONS ((CONTROLS ?AG ?C) (= (AGENT-FORCE) ?AG)) :EFFECTS ((INCREASE (CART-VELOCITY ?C) ?FORCE))
Event Model	$ev \in Ev$	$Ev \equiv S \times V \times (V \rightarrow Ty) \times [0..1] \times$ $\mathbb{R}^{\geq 0} \times Cn \times E$ Tuple: $\langle name, quals, qualTypes, prob,$ $freq, triggers, effs \rangle$	(:EVENT FINISHES :QUALITIES (?C – CART) :TRIGGERS ((UPRIGHT ?C) (< (CART-VELOCITY ?C) 0.1) (> (CART-VELOCITY ?C) -0.1) (CONTROLS ?AG ?C)) :EFFECTS (WINS ?AG))
Process Model	$p \in P$	$P \equiv S \times V \times (V \rightarrow Ty) \times Cn \times C$ Tuple:	(:PROCESS POLE-ANGLE-CHANGES :QUALITIES (?C – CART)

		$\langle name, quals, qualTypes, conds, changes \rangle$	:CONDITIONS (FORALL ?AG TRUE (NOT (WINS ?AG))) :EFFECTS (AND (INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C))))))

Table 3 – Summary of key T-SAL domain concepts with definitions and examples

In describing T-SAL domains, we use symbols as identifiers; we refer to the set of symbols as S . Variables in a domain description come from the set $V \subset S$ (the space of variable sets is denoted $\mathcal{V}$). Another set of symbols identify particular objects; we say object o comes from the set $O \subset S$.

⌈ A domain type ty is in the space $Ty \subset S$. Five symbols are reserved for special basic types: REAL, INTEGER, BOOLEAN, AGENT, and OBJECT. A sixth standard type, POSITION, is also reserved, but is not a “basic” type. Nearly every domain requires objects and sometimes agents to be related spatially, and spatial information is useful to distinguish from non-spatial. Therefore, POSITION is reserved as a type to allow spatial properties to be automatically distinguished from non-spatial properties. The POSITION type must be specified to derive from REAL, INTEGER, or OBJECT, depending on the representation of a domain.

Definition 1: Domain Types $\lrcorner$

Example: Unique types in CartPole include CART, BLOCK, SPEED, and ANGLE.

⌈ A domain function uses the variable f and has space F and space of sets $\mathbf{F}$, has a name ($function-name: F \rightarrow Fn$), arguments ($function-arguments: F \rightarrow \mathbf{V}$) and types ($function-argument-type: F \times \mathbf{V} \rightarrow Ty$), and a value type ($function-value-type: F \rightarrow Ty$). Function names fn come from the set $Fn \subset S$. In tuple form, a function is given as $\langle name, arguments, argument-types, value-type \rangle$.

Definition 2: Domain Functions and Function Names $\lrcorner$

Example: In T-SAL, the relationship between the cart and pole are described using roll, pitch, and yaw. The function f below describes the roll of a pole relative to its cart in T-SAL-CR:

$$f: (\text{CART-VELOCITY ?C} - \text{CART}) - \text{SPEED}$$

For this function, the following equivalences hold: $function\_name(f) = \text{CART-VELOCITY}$; $function\_arguments(f) = [?c]$; $function\_argument\_type(f, ?c) = \text{CART}$; $function\_value\_type(f) = \text{SPEED}$.

Values include objects, numbers, and Boolean literals; the set of values is defined $Va \equiv O \cup \mathbb{I} \cup \mathbb{R} \cup \{\text{TRUE}, \text{FALSE}\}$. Terms include values, variables, and function terms. Function terms apply a function to other terms, including function terms. Formally, the set of terms is defined $Te \equiv V \cup Va \cup \langle Fn, \mathbf{Te} \rangle$. Here, $\mathbf{Te}$ is the space of sequences of terms from Te . Legal function terms have a term sequence the same length as the arity of the named function.

Definition 3: Values and Terms $\lrcorner$

Examples: In the T-SAL domain language representation of Cart-Pole, legal terms include 30, CART1, ?C, (POLE-ANGLE ?C), and (POLE-ANGLE (LEFTMOST-CART)).

▮ Conditions (variable *cnd*, space *Cn*, space of sequences ***Cn***) specify complex requirements over states using comparisons and Boolean operators. Like calculations, conditions have the form of a term, a comparison of two values, or a combined calculation in which two operands are combined by a Boolean operation: $Cn \equiv Te \cup BOp \times Cn \times Cn \cup \text{FORALL} \times V \times Cn \times Cn \cup Ineq \times Ca \times Ca \cup \text{NOT} \times Cn$. Supported Boolean operations are $BOp \equiv \{\text{AND}, \text{OR}\}$. Supported comparison operators are the inequalities $Ineq \equiv \{=, \neq, <, >, \leq, \geq\}$.

Definition 4: Conditions ▮

Examples: In the T-SAL knowledge representation of Cart-Pole, legal conditions include:

- ($< (\text{POLE-ANGLE } ?C) 30$)
 - True iff the pole on the cart bound to ?C has roll less than 30
- ($\text{AND} (< (\text{POLE-ANGLE CART1}) 30) (> (\text{POLE-ANGLE CART1}) -30)$)
 - True iff the pole on the object CART1 has angle between -30 and 30
- ($\text{AND} (<= (\text{POLE-ANGLE CART1}) (\text{POLE-ANGLE CART2}))$)
 - True iff the pole on the object CART1 has a smaller angle than that on CART2.
- ($\text{FORALL } ?C (!= (\text{CART-VELOCITY } ?C) 0) (< (\text{POLE-ANGLE } ?C) 30)$)
 - True iff every pole with non-zero velocity also has pole angle less than 30

▮ Calculations (variable *calc*, space *Ca*, space of sets ***Ca***) are functional expressions composed of terms and operators. Calculation operators include basic arithmetic operations, distribution draws, aggregations over sets, and the trinary if operator. Where terms have the semantics of a lookup in a database incorporating state information, calculations describe executed procedures that return values. A calculation must be a term, a combined calculation in which two operands are combined by an operation, an aggregation over a set of variables, or a trinary if that returns the result of one of two sub-calculations: $Ca \equiv Te \cup Op \times Ca \times Ca \cup AggOp \times V \times Cn \times Ca \cup \text{IF } Cn \times Ca \times Ca$. Supported operations are $Op \in \{+, -, *, /, :UNIFORM, :GAUSSIAN\}$. These operations have the usual definitions, and all require that their operands are numeric (i.e., derived from the type REAL or INTEGER). The +, -, *, and / operations yield a value of type REAL if either operand is REAL, or INTEGER otherwise. The :UNIFORM operation draws a value from the uniform distribution, with a resulting value in the range defined by its operands (the first operand providing the lower bound, and the second the upper bound, inclusive). The operands and the resulting value must have the INTEGER type. The :GAUSSIAN operation draws from the gaussian distribution; its operands are interpreted as a mean and standard distribution, and the type of the calculation will be REAL. Aggregation operations are $AggOp \in (\text{SUM}, \text{PRODUCT})$. Aggregations have a single variable, constrained to meet a condition; the sub-calculation is calculated for every permissible value of the variable and results are aggregated. Conditional calculations give a condition that determines which one of a pair of calculations to evaluate; if the condition evaluates to TRUE, the first calculation is evaluated, and if FALSE the second.

Definition 5: Calculations ▮

Example: In the T-SAL knowledge representation of Cart-Pole, legal calculations include:

- (POLE-ANGLE ?C)
- (+ (POLE-ANGLE ?C) .01)
- (/ (SUM ?C ($\neq$ (CART-VELOCITY ?C) 0) (POLE-ANGLE ?C)) (SUM ?C ($\neq$ (CART-VELOCITY ?C) 0) 1))
 - Average angle of poles with nonzero velocity

⌈ Axioms define the truth of fluents in a state in terms of a condition. A fluent defined by axioms must have value type BOOLEAN and takes on the value TRUE when the associated condition is satisfied, or FALSE otherwise. No function may therefore both have an axiom definition and appear in action or event effects. Axioms use the variable ax (Ax for the space, Ax for the space of sets), and describes a function name ($axiom-name: Ax \rightarrow S$), argument types ($axiom-argument: Ax \times V \rightarrow S$), and an antecedent condition ($axiom-antecedent: Ax \rightarrow Cn$).

Definition 6: Axioms ⌋

Example: In Cart-Pole, an axiom ax could say that a “live” cart is one whose pole angles are less than 30:

(:- (UPRIGHT ?C - CART)
(AND (< (POLE-ROLL ?C) 0.1) (> (POLE-ANGLE ?C) -0.1)))

Here, $axiom-name(ax) = UPRIGHT$; $axiom-argument(ax, ?C) = CART$; and $axiom-antecedent(ax) = \langle AND, \langle <, POLE-ANGLE, [?C], 0.1 \rangle \langle >, POLE-ANGLE, [?C], -0.1 \rangle \rangle$

⌈ An effect, which uses the variable e and has space E (E for the space of sets), describes an instantaneous alteration to the world caused by an action or effect transition. Mostly, these change the values of fluents. However, a second type of effect creates a new object; this is necessary in general to allow us to describe open worlds where objects can proliferate without a bound. An effect has a symbol that names the function it affects ($effect-name: E \rightarrow Fn$), assignments to that function’s arguments ($effect-argument: E \times V \rightarrow Te$), a modification type ($effect-modification: E \rightarrow Alt$), post-effect value ($effect-value: E \rightarrow Te$), and probability ($effect-probability: E \rightarrow [0..1]$). The set of possible alterations is given by: $Alt \equiv \{SET, INCREASE, DECREASE, CREATE\}$. In the special case of a CREATE alteration, the effect’s name is the type symbol of the object created rather than that of a function, and the effect’s value describes a name prefix to use in referring to the new object.

Definition 7: Effects ⌋

Example: In Cart-Pole, an example legal effect e is (DECREASE (POLE-ANGLE ?C) 0.1) [0.5], which states that the value of the fluent POLE-ANGLE for the object bound to ?C decreases by 0.1 during a transition. Here, $effect-name(e) = POLE-ANGLE$; $effect-argument(e, ?C) = ?C$; $effect-modification(e) = DECREASE$; $effect-value(e) = 0.1$; and $effect-probability(e) = 0.5$.

A CREATE effect of the form (CREATE BLOCK ?B “new-block”) causes a new block to exist in the Cart-Pole environment. Here, $effect-name(e) = BLOCK$; $effect-argument(e, BLOCK) =$

?B; *effect-modification*(e) = CREATE; and *effect-value*(e) = “new-block”. With no probability explicitly described, we give *effect-probability*(e) = 1.

⌈ A continuous change, which uses the variable c and has space C (C for the space of sets), describes continuous time changes to fluents. A change describes the fluent it changes with a name symbol (*change-name*: $C \rightarrow S$), argument assignments to that fluent (*change-argument*: $C \times V \rightarrow Te$), direction of change (*change-direction*: $C \rightarrow Dir$) and a derivative calculation (*change-derivative*: $C \rightarrow Ca$). Here, the set of possible directions is given by: $Dir \equiv (\text{increase, decrease})$. A special DT symbol is reserved to represent the time differential in continuous change derivatives only. Each continuous change must use this symbol once.

Definition 8: Continuous Changes ┘

Example: In Cart-Pole, an example legal change c is (INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C))), which states that the value of the fluent POLE-ANGLE for the cart bound to ?C increases linearly with time according to the angular motion function (not described further here). Here, *change-name*(c) = POLE-ANGLE; *change-argument*(c , ?C) = ?C; *change-direction*(c) = INCREASE; and *change-derivative*(c) = (* DT (ANGULAR-MOTION ?C)).

⌈ Action models, event models, and process models describe when and how the world changes. Actions represent change caused by a performer; a performer can choose whether to take an action and with what parameters. Some actions may act differently for different performers or not be available to certain performers, so a performer variable gives a reference to the performer for preconditions and effects. Formally, an action model $a \in A$ (space of sets A) has a name (*action-name*: $A \rightarrow S$), performer (*action-performer*: $A \rightarrow Te$), parameters (*action-parameters*: $A \rightarrow V$), parameter types (*action-parameter-type*: $A \times V \rightarrow Ty$), preconditions (*action-preconditions*: $A \rightarrow Cn$), and effects (*action-effects*: $A \rightarrow E$). Tuple notation for actions is $\langle name, performer, args, argTypes, precs, effs \rangle$; a tuple in this form can be interpreted as an action that can respond to each above function.

Definition 9: Action Models ┘

Example: An action model a in cart pole pushes the cart forward, instantaneously increasing its velocity in the x direction:

```
(:ACTION PUSH  
:PERFORMER ?AG  
:PARAMETERS (?C – CART)  
:PRECONDITIONS ((CONTROLS ?AG ?C) (= (AGENT-FORCE ?AG) ?FORCE))  
:EFFECTS ((INCREASE (CART-VELOCITY ?C) ?FORCE))
```

Here, *action-name*(a) = PUSH; *action-performer*(a) = ?AG; *action-parameters*(a) = [?C]; *action-parameter-type*(a , ?C) = CART; *action-preconditions*(a) = [(= (AGENT-FORCE ?AG) ?FORCE) (CONTROLS ?AG ?C)]; and *action-effects*(a) = [(INCREASE (CART-VELOCITY ?C) 1)].

⌈ Event models represent instantaneous change not caused by a modelled agent; they have no parameters, because no entity (performer or otherwise) chooses an event to happen. However, qualities refer to a set of variables that may be instantiated within the triggers of the event and aid in understanding. All events are triggered in one of three ways: occur 1) by default, they must

occur immediately when their triggers are met; 2) if an event model has a probability between 0 and 1, modelled events occur probabilistically when their triggers are met; 3) if an event model has a non-zero frequency, modelled events occur at random times, with interarrival times described by a poisson distribution with the modelled frequency. No event model can have both a probability less than 1 and a non-zero frequency. An event $ev \in Ev$ (space of sets $\mathbf{Ev}$) has a name ($event-name: Ev \rightarrow S$), qualities ($event-qualities: Ev \rightarrow V$), quality types ($event-quality-type: Ev \times V \rightarrow Ty$), probability ($event-probability: Ev \rightarrow [0..1]$), frequency ($event-frequency: Ev \rightarrow \mathbb{R}^{\geq 0}$), triggers ($event-triggers: Ev \rightarrow \mathbf{Cn}$), and effects ($event-effects: Ev \rightarrow \mathbf{E}$). Tuple notation for events is $\langle name, quals, qualTypes, prob, freq, triggers, effs \rangle$; a tuple in this form can be interpreted as an event that can respond to each above function.

Definition 10: Event Models ┘

Example: An event ev in cart pole causes a win for a player whose cart reaches an angle and velocity within a certain range:

```
(:EVENT FINISHES
:QUALITIES (?C – CART)
:TRIGGERS ((UPRIGHT ?C) (< (CART-VELOCITY ?C) 0.1) (> (CART-VELOCITY ?C) -0.1)
(CONTROLS ?AG ?C))
:EFFECTS (WINS ?AG))
```

Here, $event-name(ev) = FINISHES$; $event-probability(ev) = 1$; $event-frequency(ev) = 0$; $event-qualities(ev) = [?C]$; $event-quality-type(ev, ?C) = CART$; $event-triggers(ev) = [(UPRIGHT ?C) TRUE], (< (CART-VELOCITY ?C) 0.1), (> (CART-VELOCITY ?C) -0.1), (= (CONTROLS ?AG ?C) TRUE)]$; and $event-effects(ev) = [(SET (WINS ?AG) TRUE)]$.

┐ Process models represent continuous change over time. Like events, they have qualities and are not controlled by any agent. Unlike events (or actions), they represent change over time, and describe continuous changes that occur over the duration of the process rather than effects. A process will continue for as long as its conditions are met. A process $p \in P$ (space of sets $\mathbf{P}$) has a name ($process-name: P \rightarrow S$), qualities ($process-qualities: P \rightarrow V$), quality types ($process-quality-type: P \times V \rightarrow Ty$), conditions ($process-conditions: P \rightarrow \mathbf{Cn}$), and continuous changes ($process-changes: P \rightarrow \mathbf{C}$). Tuple notation for processes is $\langle name, quals, qualTypes, conds, changes \rangle$; a tuple in this form can be interpreted as a process model that can respond to each above function.

Definition 11: Process Models ┘

Example: A process p in cart pole causes the pole to change position:

```
(:PROCESS POLE-ANGLE-CHANGES
:QUALITIES (?C – CART)
:CONDITIONS (FORALL ?AG TRUE (NOT (WINS ?AG))))
:EFFECTS (AND (INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C))))))
```

Here, $process-name(p) = POLE-ANGLE-CHANGES$; $process-qualities(p) = [?C]$; $process-quality-type(p, ?C) = CART$; $process-conditions(p) = [(FORALL ?AG TRUE (= (WINS ?AG) FALSE))]$; and $process-changes(p) = [(INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C)))]$.

⌈ A T-SAL domain, which we use the variable d for (D is the space of possible domains), includes types ($domain-types: D \rightarrow S$), supertypes ($domain-supertypes: D \rightarrow S \times S$), named constants with given types ($domain-constants: D \rightarrow S \times S$), functions ($domain-functions: D \rightarrow F$), axioms ($domain-axioms: D \rightarrow Ax$), actions ($domain-actions: D \rightarrow A$), events ($domain-events: D \rightarrow E$), and processes ($domain-processes: D \rightarrow P$). Tuple notation for domains is $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle$.

Definition 12: T-SAL Domains ⌋

Legality of a T-SAL domain, based mainly on type agreement, is described in Appendix C.

4.2.2 T-SAL States, Goals, and Scenario Generators

In addition to dynamics, T-SAL describes environments in terms of distributions of starting states and tasks. In this section, we define T-SAL scenario generators, which define both of these. To describe a scenario generator, we need the concepts of state, and various simpler generator functions: object generators, value generators, and fluent generators.

⌈ A T-SAL state st describes what is true in the environment at a particular instant in time. We describe a state with a tuple $\langle objects, defaults, assignments \rangle$. Each T-SAL state is associated with a T-SAL domain $d = \langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle$, and defines the current set of objects belonging to every type in $types$ and a value for every possible fluent grounding using $functions$ with those objects. State objects ($state-objects: St \times S \rightarrow S$) includes all objects of every type in the domain. The defaults ($state-defaults: St \times F \rightarrow Te$) give a default value for every function, which is assigned by this state to any fluent grounding that is not given a specific value in $assignments$. The state's assignments ($state-assignments: St \rightarrow GF$) give specific values for a list of ground fluents. Ground fluents use the variable gf , space GF and space of sequences GF . Each ground fluent names the function the fluent is from ($ground-fluent-function-name: GF \rightarrow Fn$), argument values ($ground-fluent-argument: GF \times V \rightarrow Va$) and a fluent value ($ground-fluent-value: GF \rightarrow Va$). Tuple form for ground fluents is $\langle function-name, argument-values, fluent-value \rangle$.

Definition 13: T-SAL State ⌋

Example: In a simple Cart-Pole state st there is one cart, CART1, and three blocks, BLOCK1, BLOCK2, and BLOCK3. A single agent, AGENT1 exists. Other types (speed, position, angle) are derived from REAL and are non-enumerated. Fluents in the domain include $f1 = (CART-VELOCITY ?C - CART) - SPEED$, $f2 = (CART-POSITION ?C - CART) - POSITION$, $f3 = (POLE-ANGLE ?C - CART) - ANGLE$, $f4 = (BLOCK-POSITION ?B - BLOCK) - POSITION$, $f5 = (BLOCK-VELOCITY ?B - BLOCK) - SPEED$, $f6 = (AGENT-FORCE ?AG - AGENT) - REAL$, $f7 = (CONTROLS ?AG - AGENT ?C - CART) - BOOLEAN$, and $f8 = (WINS ?AG - AGENT) - BOOLEAN$. Relevant values are: $state-objects(st, CART) = [CART1]$; $state-objects(st, BLOCK) = [BLOCK1, BLOCK2, BLOCK3]$; $state-defaults(st, f1) = 0$; $state-defaults(st, f3) = 0.1$; $state-defaults(st, f5) = 0$; $state-defaults(st, f6) = 1$; $state-defaults(st, f7) = FALSE$; $state-defaults(st, f8) = FALSE$; $state-assignments(st) = [CART-POSITION(CART1) = 0, BLOCK-POSITION(BLOCK1) = 5, BLOCK-POSITION(BLOCK2) = -5, BLOCK-POSITION(BLOCK3) = 10, CONTROLS(AGENT1, CART1) = TRUE]$.

Note that every legal assignment to every fluent can be valued using either a default or an explicit assignment in *st*. In this case, functions *f2* and *f4*, which have no defaults, must be fully enumerated, as shown.

▮ An object generator describes a distribution of objects in the environment. Whereas a T-SAL domain refers to certain “constant” objects present in every scenario, object generators describe the objects that are present in an environment as part of a particular scenario. An object generator *og* has a name (*object-generator-name*: $OG \rightarrow S$), type (*object-generator-type*: $OG \rightarrow S$), and a draw function that yields a set of distinct objects when called (*object-generator-draw*: $OG \rightarrow (\rightarrow S)$). As randomness is frequently a desirable property of such generators, the draw function may return sets with different contents and sizes on subsequent invocations. A set of basic draw functions is described in Section Appendix C.

Definition 14: Object Generator ▮

Example: A simple object generator *og* makes 5 block objects. We write this like so:

OBJECTGENERATOR(blockGroup, BLOCK, OBJECTLIST(5, “block”))

The standard draw function OBJECTLIST is defined to return *N* (the first argument) object names, each starting with the prefix given by the second argument. The object generator *og* has the following characteristics: *object-generator-name*(*og*) = cartGroup; *object-generator-type*(*og*) = CART; *object-generator-draw*(*og*) = OBJECTLIST(5, “cart”).

▮ A value generator returns values from a random distribution; these values may be generated by from sets defined by object generators, domain constants, or from the reals, integers, or booleans. A value generator *vg* has a name (*value-generator-name*: $VG \rightarrow S$) and provides a draw function that yields a sequence of values when called (*value-generator-draw*: $VG \rightarrow (\rightarrow Va)$). The draw function may produce random results or static results; a set of basic value generator draw functions is given in Section Appendix C.

Definition 15: Value Generator ▮

Example: A simple value generator *vg* for positions gives a single random real value between -20 and 20. We write this like so:

VALUEGENERATOR(RANDOMPOSITION, UNIFORMREALDISTRIBUTION(-20, 20))

This uses a standard draw function that pulls from a uniform distribution. The value generator *vg* has the following characteristics: *value-generator-name*(*vg*) = RANDOMPOSITION; *value-generator-draw*(*og*) = UNIFORMREALDISTRIBUTION(-20, 20) .

▮ A fluent generator describes a distribution of fluent values for a particular domain function. A fluent generator *fg* has a function name (*fluent-generator-function-name*: $FG \rightarrow Fn$) that it generates fluents for, and provides a draw function (*fluent-generator-draw*: $FG \rightarrow (\rightarrow GF)$) that yields a list of ground fluents describing individual fluent groundings; these do not necessarily map all possible assignments to the named function. Basic fluent generators are described in Section Appendix C.

Definition 16: Fluent Generator ▮

Example: A simple fluent generator fg for assigning block positions gives a random real value between -20 and 20 to each block. We write this like so:

FLUENTGENERATOR(BLOCK-POSITION, ALLPERMUTATIONS([BLOCKGROUP],
RANDOMPOSITION)

The standard draw function, ALLPERMUTATIONS, described here creates a ground fluent for every permutation of unique objects generated by the object generators given as its first argument, and assigns each a value generated by the value generator given as its second argument. The fluent generator has the following characteristics: *fluent-generator-function-name*(fg) = BLOCK-POSITION; *fluent-generator-draw*(fg) = ALLPERMUTATIONS([BLOCKGROUP], RANDOMPOSITION).

⌈ A T-SAL scenario generator sg is a tuple $\langle fluents, defaults, object-generators, value-generators, fluent-generators, performance \rangle$, that describes a distribution over environment starting states, as well as a performance calculation that describes the success of each agent. The first item, *fluents*, (*scenario-generator-object-generators*: $SG \rightarrow \mathbf{GF}$), gives a set of ground fluents present in every scenario generated; these are not randomized. The second item, *defaults*, (*scenario-generator-defaults*: $SG \times Fn \rightarrow Va$) maps each function name to a default value that can be assumed in the absence of contradictory assignments (from the fluents list and fluent generators). Each fluent in the corresponding domain should be covered by this mapping. The third item, *object-generators*, (*scenario-generator-object-generators*: $SG \rightarrow \mathbf{OG}$) sets up objects present in a starting state. The fourth item, *value-generators*, (*scenario-generator-value-generators*: $SG \rightarrow \mathbf{VG}$) gives a set of value generators defined for use by the fluent generators. The fifth scenario generator item, *fluent-generators*, (*scenario-generator-fluent-generators*: $SG \rightarrow \mathbf{FG}$) gives a set of fluent generators that describe distributions over fluents in the starting state. Finally, the last item, *performance*, gives a performance calculation as a calculation (*scenario-generator-performance-calculation*: $SG \rightarrow Ca$). This calculation must have a single free variable, ?AG, which identifies the agent for which performance is to be computed. For simplicity, we assume that this calculation is always intended to be maximized and can be evaluated in any state.

Definition 17: T-SAL Scenario Generator ⌋

Example: A scenario generator sg defines cart-pole performance as the negative sum of the squared angles of all poles controlled by that agent.

scenario-generator-performance-calculation(sg) =
(SUM ?C (CONTROLS ?AG ?C) (- (* (POLE-ANGLE ?C) (POLE-ANGLE ?C))))

To generate a scenario, a new state object is created with defaults from the state generator, objects created by drawing from each object-generator, and assignments created by drawing from each fluent generator. Value generators are used indirectly during fluent generators draws. Finally, fluents in the scenario generator fluents list are added to the state, overwriting any generated and/or default fluents with the same fluent name and arguments. An environment consists of a domain d and a scenario generator sg . To be legal, sg must have a close relationship to the domain d :

- *fluents* must have only ground fluents with appropriate arity and values of appropriate types
- *defaults* must give an appropriately typed value for every function in d
- *object-generators* must use object types in d

- *fluent-generators* must always generate legal ground fluents for the described function, which must also be in domain d
- *performance* must be a real-valued calculation with a single free variable, $?AG$, referring to the target agent, it must only reference functions in the domain d , and all symbols must reference constants from the domain d .

Since the returns from generator draw functions are defined to vary over time, constraints on the draw function outputs are expressed below using the “G” temporal operator, which means “always”.

4.2.3 T-SAL Transformations and Transformation Sequences

⌈ We use the variable t to describe an environment transformation, given as a tuple with a head and named arguments. The space of possible environment transformations is denoted T . We will also refer to a transformation sequence $\mathbf{t} = [t_0, \dots, t_n]$, with the transformation sequence space denoted as $\mathbf{T}$. A transformation has a type (*transformation-type*: $T \rightarrow S$) and arguments (*transformation-argument*: $T \times S \rightarrow U$). Here, U denotes the universal set; the meaning of particular transformation arguments is dependent on the type. Individual transformations are defined in Appendix C (T-Transformations) and Appendix C (R-Transformations).

Definition 18: Transformations and Transformation Sequences

Example: A transformation t to a cart-pole environment that subtracts push force from a budget is given as:

ADDACTIONEFFECT(ACTIONNAME: PUSH, EFFECT: (DECREASE (PUSH-BUDGET ?AG)
?FORCE))

The transformation t has *transformation-type*(t) = ADDACTIONEFFECT. The first argument is referred to by *transformation-argument*(t , ACTIONNAME) = PUSH.

⌈ Applying a transformation to an environment yields a new environment: *apply*: $T \times D \times SG \rightarrow D \times SG$. We also define a function that applies a sequence of transformations (*applySequence*: $\mathbf{T} \times D \times SG \rightarrow D \times SG$) as follows.

$$\begin{aligned} \text{applySequence}(\emptyset, d, sg) &\equiv \langle d, sg \rangle \\ \text{applySequence}(\mathbf{t} = [t_0, \dots, t_n], d, sg) &\equiv \text{applySequence}([t_1, \dots, t_n], \text{apply}(t_0, d, sg)) . \end{aligned}$$

Definition 19: Transformation and Transformation Sequence Application

4.2.4 T-SAL R-Transformations

R-transformations are modifications to the environment dynamics. We formally define 30 transformations on T-SAL environments in Appendix A with a summary of these given here in Table 4. State space transformations are those that change the set of states the environment may contain. Performer control transformations change a performer’s ability to interact with the environment. Instantaneous environment change transformations change all non-voluntary instantaneous transitions. Durative natural change transformations change processes which operate over continuously over time.

Summary of all R-transformations			
State Space Transformations	Performer Control Transformations	Instantaneous Environment Change Transformations	Durative Natural Change Transformations
ADDDTYPE	ADDACTION	ADDEVENT	ADDPROCESS
ADDDTYPEPARENT	ADDPRECONDITION	CHANGEFREQUENCY	ADDPROCESSCONDITION
ADDCONSTANT	ADDACTIONEFFECT	CHANGEPROBABILITY	ADDPROCESSCHANGE
ADDFUNCTION	REMOVEACTION	ADDTRIGGER	REMOVEPROCESS
ADDAXIOM	REMOVEPRECONDITION	ADDEVENTEFFECT	REMOVEPROCESSCONDITIONS
REMOVETYPE	REMOVEACTIONEFFECT	REMOVEEVENT	REMOVEPROCESSCHANGE
REMOVETYPEPARENT		REMOVETRIGGER	
REMOVECONSTANT		REMOVEEVENTEFFECT	
REMOVEFUNCTION			
REMOVEAXIOM			

Table 4: All R-transformations. The *apply* function is defined for each R-Transformations in Appendix C.

4.2.5 T-SAL T-Transformations

T-Transformations are modifications to the scenario generator. We define a set of T-Transformations which cover the space of possible generator modifications below; first, we give examples for a paradigmatic domain, then we define the particular transformations possible, as well as the sub-generators that can be used.

To easily transform scenario generators, it will be useful to supersede existing value or object generators. Therefore, when we add a new generator, it should be assumed to replace any existing generator with the same name, and all references in other generators to the old generator become references to the new generator. As such, we need only five types of T-Transformation:

ADDFLUENTVALUE, ADDDEFAULTVALUE, ADDOBJECTGENERATOR, ADDVALUEGENERATOR, ADDFLUENTGENERATOR, and ADDPERFORMANCECALCULATION. As they are particularly novel, we give additional examples below of T-Transformations for value, object, and fluent generators useful in various domains; we follow this with a formal definition of these five transformations, and a description of the value, object, and fluent generators used.

4.2.5.1 T--Transformations Examples

In Mudworld (Molineaux and Aha 2014), a set of rovers attempt to travel to a set of destinations on a 6x6 grid. Any location may be muddy, with a 30% chance. Mud slows the rover down to half speed, with some variability across scenarios. All destinations are within four cell moves. The following T-Transformations construct a scenario generator for this environment.

```
ADDVALUEGENERATOR(XLoc, UNIFORMINTEGERDISTRIBUTION(1, 6))
ADDVALUEGENERATOR(Yloc, UNIFORMINTEGERDISTRIBUTION(1, 6))
ADDVALUEGENERATOR(Xset, INTEGERSEQUENCE(1, 6))
ADDVALUEGENERATOR(Yset, INTEGERSEQUENCE(1, 6))
ADDOBJECTGENERATOR(robotGroup, Robot, OBJECTLIST(6, "rover"))
ADDVALUEGENERATOR(RobotG, DRAWFROMOBJECTSET(robotGroup))
```

```
ADDVALUEGENERATOR(RandRobot, DRAWALLFROMOBJECTSET(robotGroup))
ADDVALUEGENERATOR(TRUE, CONSTANTFUNCTION(TRUE))
ADDVALUEGENERATOR(IsMuddy, BERNOULLIDISTRIBUTION(0.3))
ADDFLUENTGENERATOR("muddy", ALLPERMUTATIONS([Xset, Yset], IsMuddy))
ADDVALUEGENERATOR(SpeedInMud, GAUSSIANDISTRIBUTION(0.5, 0.2))
ADDFLUENTGENERATOR("speed-in-mud", ALLPERMUTATIONS([], SpeedInMud))
ADDVALUEGENERATOR(ClosePos, FILTER(
    DRAWTUPLE([Xloc, Yloc, Xloc, Yloc], ["x1", "y1", "x2",
    "y2"])),
     $\lambda(x_1, y_1, x_2, y_2) \rightarrow \{|x_1 - x_2| + |y_1 - y_2| < 4\}$ )))
ADDVALUEGENERATOR(CloseSet, NEWSET(NDRAWS(ClosePos, 6))
ADDFLUENTGENERATOR("robot-x-loc", NFLUENTDRAWS([Robot], CloseSet.x1, 6))
ADDFLUENTGENERATOR("robot-y-loc", NFLUENTDRAWS([Robot], CloseSet.y1, 6))
ADDFLUENTGENERATOR("robot-dest", NFLUENTDRAWS([Robot, CloseSet.x2, CloseSet.y2],
TRUE, 6))
```

Here, we consider a scenario generator that creates a multi-dimensional random value for a camera's starting location:

```
ADDVALUEGENERATOR(Position,
    NDIMENSIONALGAUSSIANDISTRIBUTION([0.5, 0.7, 0.2], [0.1, 0.2, 0.05]))
ADDFLUENTGENERATOR("camera-starting-position",
    NFLUENTDRAWS([Position.x1, Position.x2, Position.x3], True, 1))
```

4.2.5.2 T--Transformations Examples

We define the apply function for five T-Transformations:

```
apply(ADDFLUENTVALUE(FUNCTIONNAME: name, FLUENTARGS: args, VALUE: value), d,
    ⟨fluents, defaults, object-generators, value-generators, fluent-generators, performance⟩)
≡
    ⟨{⟨fname, fargs, fvalue⟩ | ⟨fname, fargs, fvalue⟩ ∈ fluents ∧ (fname ≠ name ∨ fargs ≠ args)}
    ∪ {⟨name, args, value⟩},
    defaults, object-generators, value-generators, fluent-generators⟩
apply(ADDDEFAULTVALUE(FUNCTIONNAME: name, VALUE: value), d,
    ⟨fluents, defaults, object-generators, value-generators, fluent-generators, performance⟩)
≡
    ⟨fluents, {⟨fname, fvalue⟩ | ⟨fname, fvalue⟩ ∈ defaults ∧ fname ≠ name} ∪ {⟨name, value⟩},
    object-generators, value-generators, fluent-generators⟩
apply(ADDOBJECTGENERATOR (NAME: name, TYPE: type, DRAWFUNCTION: drawFn), d,
    ⟨fluents, defaults, object-generators, value-generators, fluent-generators, performance⟩)
≡
    ⟨fluents, defaults,
    {og ∈ object-generators | object-generator-name(og) ≠ object-generator-
    name(newGenerator)}⟩
```

$$\cup \{\langle name, type, drawFn \rangle\},$$
$$value-generators, fluent-generators\}$$
$$apply(ADDVALUEGENERATOR(NAME: name, DRAWFUNCTION: drawFn), d,$$
$$\langle fluents, defaults, object-generators, value-generators, fluent-generators, performance \rangle)$$
$$\equiv$$
$$\langle fluents, defaults, object-generators,$$
$$\{vg \in value-generators \mid value-generator-name(vg) \neq value-generator-$$
$$name(newGenerator)\}$$
$$\cup \{\langle name, drawFn \rangle\}, fluent-generators\}$$
$$apply(ADDFLUENTGENERATOR(NAME: name, DRAWFUNCTION: drawFn), d,$$
$$\langle fluents, defaults, object-generators, value-generators, fluent-generators, performance \rangle)$$
$$\equiv$$
$$\langle fluents, defaults, object-generators, value-generators,$$
$$\{fg \in fluent-generators \mid fluent-generator-name(fg) \neq fluent-generator-$$
$$name(newGenerator)\}$$
$$\cup \{\langle name, drawFn \rangle\}\}$$
$$apply(REPLACEPERFORMANCECALCULATION(PERFORMANCE: calculation), d,$$
$$\langle fluents, defaults, object-generators, value-generators, fluent-generators, performance \rangle)$$
$$\equiv$$
$$\langle fluents, defaults, object-generators, value-generators, fluent-generators, calculation \rangle$$

4.2.6 Object Generator Draw Functions

The following simple example of an object generator draw function grounds our examples. Others, particularly with variable output, are possible.

$OBJECTLIST(count, id) = () \rightarrow [gensym(id, i)]_{i=1..count}$

Generates *count* objects of the requested *type*, prepended with the string in *id*, as created by the Lisp “gensym” function.

4.2.7 Value Generators

The following examples of value generator draw functions ground our examples. Others are possible.

$UNIFORMDISTRIBUTION(min, max)$

Generator always returns a sequence containing a single real number drawn from a uniform distribution between min and max.

$UNIFORMINTEGERDISTRIBUTION(min, max)$

Generator always returns a sequence containing a single uniform integer draw between min and max (inclusive).

$GAUSSIANDISTRIBUTION(mean, stdev)$

Generator always returns a sequence containing a single draw from a one-dimensional Gaussian distribution with the given parameters.

NDIMENSIONALGAUSSIANDISTRIBUTION(*means*[], *stdevs*[])

Generator always returns a sequence containing a single draw from an n-dimensional Gaussian distribution with the given parameters, as a tuple with values named [x1, x2, ...].

BERNOULLIDISTRIBUTION(*p*)

Generator returns TRUE with probability *p*, and otherwise returns FALSE.

INTEGERSEQUENCE(*min*, *max*)

Generator always returns a sequence containing all integers between min and max, in order.

DRAWFROMOBJECTSET(*objectGeneratorName*)

Generator always returns a sequence containing a single draw from an object generator on the same scenario generator, referenced by name.

DRAWALLFROMOBJECTSET(*objectGeneratorName*)

Generator always returns a sequence containing all objects that were produced from an object generator on the same scenario generator, referenced by name, in a random order.

CONSTANTFUNCTION(*value*)

Generator always returns a sequence containing the single value passed in, i.e., the sequence [*value*].

NDRAWS(*valueGenerator*, *n*)

Generator returns a sequence of *n* values drawn from *valueGenerator*.

DRAWTUPLE(*valueGenerators*[], *names*[])

Generator returns a sequence with a single tuple with values drawn from *valueGenerators* and addressable by names given in *names*.

FILTER(*valueGenerator*, *filterFn* (Any) → Boolean)

Generator returns a sequence with a single value; this is the first value returned by repeatedly drawing from *valueGenerator* until the *filterFn* reports TRUE for one.

NEWSET(*valueGenerator*)

Generator always returns the same sequence, which is first populated by drawing from *valueGenerator*.

4.2.8 Fluent Generators

The following examples of fluent generator draw functions ground our examples. Others are possible.

ALLPERMUTATIONS(*argumentGenerators*[], *fluentValueGenerator*)

Generator returns ground fluents with arguments populated by value generators corresponding to the names in *argumentGenerator*, and values populated by value generators corresponding to *fluentValueGenerator* (can be referenced by name or provide draw function directly).

All permutations of a set of draws from the argument generators are used to generate a list of ground fluents. Values from *fluentValueGenerator* will be used in order, and a new set drawn when none are left, until every permutation is assigned a fluent value.

NFLUENTDRAWS(*argumentGenerators*[], *fluentValueGenerator*, *n*)

Generator returns *n* ground fluents. The arguments of each are populated by value generators corresponding to the names in *argumentGenerators*, and the value of each is populated by the value generator corresponding to *fluentValueGenerator*. All values from a single draw of each argument and fluent value generator are used in order; if fewer than *n* values are returned by a generator, it will be called again until *n* values have been generated.

COMBINEFUNCTIONS(*fluentGenerators*[])

Results of each fluent generator are concatenated.

4.3 Application to Novelty Hierarchy

This section gives a novelty type definition for each level of the novelty hierarchy introduced by the DARPA SAIL-ON program (see Table 5). These definitions were an early attempt to categorize transformations possible in open-world environments, starting with the properties of individual world elements (objects, agents, and their actions and goals), advancing to multi-element properties (relations, interactions, and events), and ending with general properties broadly impacting multiple world elements (environments). Levels in the novelty hierarchy were intended to be mutually exclusive. Novelties of varying difficulty can arise from each level of the novelty hierarchy, though different novelty-handling techniques and more sophisticated world models may be required at higher numbered levels (Novelty Working Group, 2023). Note that the

Tier	Category	Definition	Notes
Single Entities	1 – Objects	An entity in the environment that is not the point-of-view agent and does not have goal-oriented behavior.	Objects may experience state changes as a result of actions by the i) point-of-view agent, ii) an external agent, iii) an event, iv) or a non-volitional behavior.
	2 – Agents	An entity in the environment that is not the point-of-view agent and has goal-oriented behavior. (People will disagree on what goal-oriented means, that is fine.)	Same as #1, except must have a goal. Goal does not have to be obvious. Random exploration can be a goal.
	3 – Actions	A goal-oriented behavior of an external agent that is not the point-of-view agent.	Goal does not have to be obvious. Random exploration can be a goal. Changes in movements and state-change attributes (a pendulum) of non-volitional entities (objects) are NOT considered actions.
Multiple Entities	4 – Relations	Static property of the relationship between multiple entities.	Can be spatial, temporal, or other. Can include any combination of entities (objects and agents) and can include the point-of-view agent.
	5 – Interactions	Dynamic property of behaviors or actions that impacts multiple entities.	Can be spatial, temporal, or other. Can include any combination of entities (objects and agents) and can include the point-of-view agent.
Complex Phenomena	6 – Environments	A change in an element of an open-world space that may impact the entire task space and is independent of a specific entity.	Can be general across the entire task space (e.g., global temperature change) or varied in space and time (e.g., partial or temporary darkness or wind).
	7 – Goals	A change in the purpose of goal-oriented behavior by an agent in the environment that is not the point-of-view agent.	If a domain already has team members of the AI agent (embodied or non-embodied) that alter goals, then changes in team or AI agent goal (directives) communicated by this external team member are in-scope.
	8 – Events	A state change or series of state changes that are not each the result of volitional action by an agent.	This includes state changes with specific preconditions.

Table 5. Novelty Hierarchy, ca. January 2022 (Kildebeck et al, 2022)

term “hierarchy” is used here for historical reasons; no hierarchical relationship is required or expected between these novelty characterizations.

The formalizations provided here show that the T-SAL R-Transformations and T-Transformations are sufficient for formalizing the novelty categorizations created for the SAIL-ON program. Future research should attempt to investigate what classes of novelty are difficult or easy for agents with various capabilities to reason about.

This section is organized as follows: there are eight novelty hierarchy levels. For each level, we first provide a natural language definition and clarifications shared in the DARPA SAIL-ON program (Kildebeck et al, 2022) to guide those groups responsible for generating novelty in these categories as to what novelties are admissible in each category. For each of the eight novelty hierarchy levels, we then provide a formal definition, first described in plain language then in L(T-SAL). The L(T-SAL) description of each level provides a decision function of the form *has<X>Novelty(t, d)*, where “<X>” is the novelty type, *t* is a transformation sequence and *d* a T-SAL domain. This function decides whether the novelty described by *t* belongs to category X. Finally, for each level we provide one or more T-SAL-CR examples of the category as transformation sequences.

The ambiguity in the natural language definitions described below is such that novelties generated by various groups adhered to different standards (NWG, 2023). It is our hope that the formal standards presented here will help future researchers to adhere to common standards.

4.3.1 Objects

Natural language definition: An entity in the environment that does not have goal-oriented behaviors.

Clarifications: Objects may experience state changes as a result of actions by i) the point-of-view agent, ii) an external agent, or iii) an event. This level is explicitly differentiated from “instance” novelty, which arises when new objects of an existing type are observed.

Formal definition: To introduce object novelty, a transformation sequence must either 1) add a new type that inherits from OBJECT, 2) add a new function that relates that object type to something else, or 3) change the initial distribution of such a function (via a new fluent generator). In addition, to ensure relevance, the new type or function must impact domain dynamics, through the preconditions of an action model, the conditions of a process model, and/or the triggers of an event model.

hasObjectNovelty(t, d, sg) ≡
$$\begin{aligned} &\exists i, f, j, p, c, d_N: 0 \leq i < |t| \\ &\wedge d_N = \text{applySequence}(t, d) \\ &\wedge (((\text{transformation-type}(t_i) = \text{ADDTYPE} \\ &\quad \vee \text{transformation-type}(t_i) = \text{ADDTYPEPARENT} \\ &\quad \vee \text{transformation-type}(t_i) = \text{REMOVETYPEPARENT}) \\ &\wedge p = \text{transformation-argument}(t_i, \text{PARENT}) \\ &\wedge c = \text{transformation-argument}(t_i, \text{CHILD}) \\ &\wedge f \in \text{domain-functions}(d) \\ &\wedge \text{derived-from}(c, \text{OBJECT}, d_N) \end{aligned}$$

$$\begin{aligned}
& \wedge \text{derived-from}(p, \text{function-argument-types}(f)_j, d) \\
& \wedge \neg \text{derived-from}(c, \text{function-argument-types}(f)_j, d) \\
& \wedge \text{relevant-function}(\text{function-name}(f), d) \\
& \vee (\text{transformation-type}(t_i) = \text{ADDFUNCTION} \wedge f = \text{transformation-argument}(t_i, \\
& \text{FUNCTION}) \\
& \quad \wedge \text{derived-from}(\text{function-argument-types}(f)_j, \text{OBJECT}) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d_N)) \\
& \vee (\text{transformation-type}(t_i) = \text{ADDFLUENTGENERATOR} \\
& \quad \wedge f = \text{fluent-generator-function}(\text{transformation-argument}(t_i, \\
& \text{FLUENTGENERATOR})) \\
& \quad \wedge \text{derived-from}(\text{function-argument-types}(f)_j, \text{OBJECT}) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d_N))
\end{aligned}$$

Required Definitions:

The relation *derived-from* is defined in Appendix C.

$$\begin{aligned}
& \text{relevant-function}(fn, d) \equiv \\
& \quad (\exists cnd: ((\exists a \in \text{domain-actions}(d) \wedge cnd \in \text{action-preconditions}(a)) \\
& \quad \vee (\exists ev \in \text{domain-events}(d) \wedge cnd \in \text{event-triggers}(ev)) \vee \\
& \quad \vee (\exists p \in \text{domain-processes}(d) \wedge cnd \in \text{process-conditions}(ev)) \\
& \quad \wedge (\text{function-in-condition}(fn, cnd)) \\
& \quad \vee \exists ax: \text{function-affects-axiom}(fn, ax, d) \wedge \text{function-in-condition}(fn, cnd)) \\
& \quad \vee \text{function-affects-performance}(fn, d)) \\
& \text{function-affects-performance}(fn, d) \equiv \\
& \quad \text{function-in-calculation}(fn, \text{domain-performance-function}(d)) \\
& \text{function-affects-axiom}(fn, ax, d) \equiv \\
& \quad \text{function-in-condition}(fn, \text{axiom-condition}(ax), d) \\
& \quad \vee \exists ax_M: ax_M \in \text{domain-axioms}(d) \\
& \quad \quad \wedge \text{function-affects-axiom}(fn, ax_M, d) \\
& \quad \quad \wedge \text{function-in-condition}(\text{axiom-name}(ax_M), \text{axiom-condition}(ax), d) \\
& \text{function-in-term}(fn, te \in Te) \equiv \perp \\
& \text{function-in-term}(fn, te = \langle f, \mathbf{te} \rangle \in F \times \mathbf{Te}) \equiv \\
& \quad \text{function-name}(f) = fn \\
& \quad \vee (\exists i: 0 \leq i < |\mathbf{te}| \wedge \text{function-in-calculation}(fn, \mathbf{te}_i)) \\
& \text{function-in-condition}(fn, cnd \in Te) \equiv \\
& \quad \text{function-in-term}(fn, cnd) \\
& \text{function-in-condition}(fn, \langle bop, cnd1, cnd2 \rangle \in bOp \times Cn \times Cn) \equiv \\
& \quad \text{function-in-condition}(fn, cnd1) \\
& \quad \vee \text{function-in-condition}(fn, cnd2) \\
& \text{function-in-condition}(fn, \langle \text{FORALL}, \mathbf{v}, cnd1, cnd2 \rangle) \equiv \\
& \quad \text{function-in-condition}(fn, cnd1) \\
& \quad \vee \text{function-in-condition}(fn, cnd2)
\end{aligned}$$

$function-in-condition(fn, \langle ineq, calc1, calc2 \rangle \in Ineq \times Ca \times Ca) \equiv$
 $function-in-calculation(fn, calc1)$
 $\vee function-in-calculation(fn, calc2)$

$function-in-calculation(fn, calc \in Te) \equiv$
 $function-in-term(fn, calc)$

$function-in-calculation(fn, \langle op, calc1, calc2 \rangle \in Op \times Ca \times Ca) \equiv$
 $function-in-calculation(fn, calc1) \vee function-in-calculation(fn, calc2)$

$function-in-calculation(fn, \langle aggo, v, con, calc \rangle \in AggOp \times V \times Cn \times Ca) \equiv$
 $function-in-condition(fn, con) \vee function-in-calculation(fn, calc)$

$function-in-calculation(fn, \langle IF, cnd, calc1, calc2 \rangle \in IF \times Cn \times Ca \times Ca) \equiv$
 $function-in-condition(fn, cnd) \vee function-in-calculation(fn, calc1) \vee function-in-$
 $calculation(fn, calc2)$

$function-argument-types(f) = [function-argument-type(f, function-argument(f)_i)]_{i=1..|function-argument(f)|}$

$fluent-generator-function(fg, d) = \operatorname{argmin}_f f \in \operatorname{domain-functions}(d)$
 $\wedge function-name(f) = fluent-generator-function-$
 $name(fg)$

Example: Every cart starts with a block within 2 distance units (3 carts standard):

```
ADDVALUEGENERATOR(NEARBY-STARTS, FILTER(  
    DRAWTUPLE([POSITION, POSITION], ["cart",  
    "block"])),  
     $\lambda(x, y) \rightarrow \{|x - y| < 2\}$ )  
ADDVALUEGENERATOR(START-POSITIONS, NEWSET(NDRAWS(NEARBY-STARTS, 3))  
ADDVALUEGENERATOR(CLOSE-BLOCKS, NDRAWS(BLOCKGROUP, 3))  
ADDVALUEGENERATOR(FAR-BLOCKS, DIFFERENCE(BLOCKGROUP, CLOSE-BLOCKS))  
ADDFLUENTGENERATOR(CART-POSITION,  
    ALLPERMUTATIONS([CARTGROUP], START-POSITION.CART)  
  
ADDFLUENTGENERATOR(BLOCK-POSITION,  
    COMBINEFUNCTIONS(ALLPERMUTATIONS([CLOSE-BLOCKS], START-  
    POSITION.BLOCK)  
    ALLPERMUTATIONS([FAR-BLOCKS], POSITION))
```

Example: Initial block velocities are doubled:

```
Original generator:  
FLUENTGENERATOR(BLOCK-VELOCITY, ALLPERMUTATIONS([BLOCKGROUP],  
    RANDOMINSET([1, -1])  
ADDFLUENTGENERATOR(BLOCK-VELOCITY, ALLPERMUTATIONS([BLOCKGROUP],  
    RANDOMINSET([2, -2])
```

4.3.2 Agents

Natural language definition: An entity in the environment that is not the point-of-view agent and has goal-oriented behavior. The agent is most frequently an external agent but can in some instances be the point-of-view agent. The goal of the agent does not have to be obvious.

Clarifications: Researchers will disagree on what ‘goal-oriented behavior’ means. This is fine, and researchers have flexibility within reason to determine best practices in specific domains. The goal of the agent does not have to be obvious. For example, random exploration could be a goal.

Formal definition: An agent-level novelty is defined identically to an object-level novelty, but using AGENT subtypes instead of OBJECT subtypes. Note that agents are primarily distinguished from objects in that every action has a performer which must be an agent.

hasAgentNovelty (t, d) $\equiv$
 $\exists i, f, j, p, c, d_N: 0 \leq i < |t|$
 $\wedge d_N = \text{applySequence}(t, d)$
 $\wedge (((\text{transformation-type}(t_i) = \text{ADDTYPE}$
 $\quad \vee \text{transformation-type}(t_i) = \text{ADDTYPEPARENT}$
 $\quad \vee \text{transformation-type}(t_i) = \text{REMOVETYPEPARENT})$
 $\wedge p = \text{transformation-argument}(t_i, \text{PARENT})$
 $\wedge c = \text{transformation-argument}(t_i, \text{CHILD})$
 $\wedge f \in \text{domain-functions}(d)$
 $\wedge \text{derived-from}(c, \text{AGENT}, d_N)$
 $\wedge \text{derived-from}(p, \text{function-argument-types}(f)_j, d)$
 $\wedge \neg \text{derived-from}(c, \text{function-argument-types}(f)_j, d)$
 $\wedge \text{relevant-function}(\text{function-name}(f), d))$
 $\vee (\text{transformation-type}(t_i) = \text{ADDFUNCTION} \wedge f = \text{transformation-argument}(t_i,$
FUNCTION)
 $\quad \wedge \text{derived-from}(\text{function-argument-types}(f)_j, \text{AGENT})$
 $\quad \wedge \text{relevant-function}(\text{function-name}(f), d_N))$
 $\vee (\text{transformation-type}(t_i) = \text{ADDFLUENTGENERATOR}$
 $\quad \wedge f = \text{fluent-generator-function}(\text{transformation-argument}(t_i,$
FLUENTGENERATOR))
 $\quad \wedge \text{derived-from}(\text{function-argument-types}(f)_j, \text{AGENT})$
 $\quad \wedge \text{relevant-function}(\text{function-name}(f), d_N))$

Required Definitions:

The relation *derived-from* is defined in Appendix C.

Example: A transformation sequence changes the amount of force other agents use when pushing, referenced by the push action (see Appendix C).

ADDVALUEGENERATOR(OTHER-AGENTS, NEWSET(DIFFERENCE(AGENTS, [AGENT1])))
ADDFLUENTGENERATOR(AGENT-FORCE, ALLPERMUTATIONS([OTHER-AGENTS],
UNIFORMDISTRIBUTION(0.5,
1.5)))

4.3.3 Actions

Natural language definition: A goal-oriented behavior of an external agent that is not the point-of-view agent.

Clarifications: Non-goal-oriented movements of non-volitional objects are not considered actions.

Formal definition: For a transformation sequence to exhibit action novelty, the preconditions or effects of an action model must be changed. In addition, we require that the performer of the changed action cannot be of the same type ty_{AG} as the point-of-view agent.

hasActionNovelty ($\mathbf{t} = [t_0 \dots t_n]$, ty_{AG} , d) =
 $\exists i < |\mathbf{t}|, d_p, d_T, a_T, n, v_T, \theta, \mathbf{v}_{BOUND}$:
 $d_p = \text{applySequence}([t_0 \dots t_{i-1}], d)$
 $\wedge \exists a \in \text{domain-actions}(d_p): n = \text{action-name}(a)$
 $\wedge \text{transformation-argument}(t_i, \text{ACTIONNAME}) = n$
 $\wedge \text{transformation-type}(t_i) \in \{\text{ADDPRECONDITION}, \text{REMOVEPRECONDITION},$
 $\text{ADDACTIONEFFECT}, \text{REMOVEACTIONEFFECT}\}$
 $\wedge d_T = \text{applySequence}(\mathbf{t}, d)$
 $\wedge a_T \in \text{domain-actions}(d_T) \wedge \text{action-name}(a_T) = n$
 $\wedge \text{action-performer}(a_T) = v_T$
 $\wedge \mathbf{v}_{BOUND} = \text{bound-variables-in-conditions}(\text{action-preconditions}(a_T), \mathbf{v}_{BOUND})$
 $\cup (\{\text{action-performer}(a_T)\} \cap V) \cup (\text{action-parameters}(a_T)$
 $\wedge \neg \exists \theta: (\text{assignable-to}(ty_{AG}, v_T, \theta, d_T)$
 $\wedge \forall cnd \in \text{action-preconditions}(a_T): \text{legal-condition}(cnd, \theta, \mathbf{v}_{BOUND}, d))$

Required Definitions:

The relation *bound-variables-in-conditions* and *assignable-to* are defined in Appendix A.

Example: An environment transformation adds an action that lets agents “donate” cart velocity to blocks

ADDACTION(DONATE, ?AG, [?C], [CART])
ADDPRECONDITION(DONATE, (= (CONTROLS ?AG ?C) TRUE))
ADDPRECONDITION(DONATE, (= (CART-VELOCITY ?C) ?V))
ADDACTIONEFFECT(DONATE, (SET (CART-VELOCITY ?C) 0))
ADDACTIONEFFECT (DONATE, (INCREASE (BLOCK-VELOCITY ?B) ?V))

Example: Cart pushes have a budget

ADDFUNCTION(PUSH-BUDGET)
ADDPRECONDITION(PUSH, (> (PUSH-BUDGET ?AG) ?FORCE))
ADDACTIONEFFECT (PUSH, (DECREASE (PUSH-BUDGET ?AG) ?FORCE))
ADDFLUENTGENERATOR(PUSH-BUDGET, ALLPERMUTATIONS([AGENTS],
CONSTANTFUNCTION(100)))

4.3.4 Relations

Natural language definition: Static properties of the relationships between multiple entities.

Clarifications: Can be spatial, temporal, or other. Can include the point-of-view agent and other entities.

Formal definition: A transformation sequence exhibits relation novelty if a static function (one not present in any effects or process changes) is created, has its distribution changed, or is added to triggers, conditions, or preconditions. The function must be relevant and must involve 2 or more entities (here, entities are specifically objects of types AGENT, OBJECT, or their subtypes).

hasRelationNovelty(*t*, *d*) =

$$\begin{aligned} & \exists f, \text{ name}, i: 0 \leq i < |t| \\ & \quad \wedge ((\text{transformation-type}(t_i) = \text{ADDFUNCTION} \\ & \quad \quad \wedge \text{function-name}(\text{transformation-argument}(t_i, \text{FUNCTION})) = \text{name}) \\ & \quad \vee (\text{transformation-type}(t_i) \in \{\text{ADDPRECONDITION}, \text{REMOVEPRECONDITION}\} \\ & \quad \quad \wedge \text{function-in-condition}(\text{name}, \text{transformation-argument}(t_i, \\ & \text{PRECONDITION}))) \\ & \quad \vee (\text{transformation-type}(t_i) \in \{\text{ADDTRIGGER}, \text{REMOVEDTRIGGER}\} \\ & \quad \quad \wedge \text{function-in-condition}(\text{name}, \text{transformation-argument}(t_i, \text{TRIGGER}))) \\ & \quad \vee (\text{transformation-type}(t_i) \in \{\text{ADDPROCESSCONDITION}, \\ & \text{REMOVEPROCESSCONDITION}\} \\ & \quad \quad \wedge \text{function-in-condition}(\text{name}, \text{transformation-argument}(t_i, \text{CONDITION}))) \\ & \quad \vee (\text{transformation-type}(t_i) = \text{ADDFLUENTGENERATOR} \\ & \quad \quad \wedge \text{name} = \text{fluent-generator-function-name}(\text{transformation-argument}(t_i, \\ & \text{FLUENTGENERATOR}))) \\ & \quad \wedge d_T = \text{applySequence}(t, d) \\ & \quad \wedge f \in \text{domain-functions}(d_T) \\ & \quad \wedge \text{function-name}(f) = \text{name} \\ & \quad \wedge |\text{entities-in-function}(f)| > 1 \\ & \quad \wedge (\forall a \in \text{domain-actions}(d_T) : \nexists e \in \text{action-effects}(a) : \text{effect-name}(e) = \text{name} \\ & \quad \quad \vee \forall ev \in \text{domain-events}(d_T) : \nexists e \in \text{event-effects}(ev) : \text{effect-name}(e) = \text{name} \\ & \quad \quad \vee \forall p \in \text{domain-processes}(d_T) : \nexists c \in \text{process-changes}(p) : \text{change-name}(c) = \\ & \text{name}) \end{aligned}$$

Required Definitions:

entities-in-function(*f*) =

$$\{v \in \text{function-arguments}(f) \mid \text{derived-from}(\text{function-argument-type}(f, v), \text{OBJECT}) \vee \text{derived-from}(\text{function-argument-type}(f, v), \text{AGENT})\}$$

Example: A transformation sequence asserts a minimum distance between every block and cart. This adds a new binary static function, MIN-DIST. The revised scenario generator assigns a random value between 0 and 1 to this function for each pair of block and cart. The PUSH action preconditions change to enforce this constraint.

$$\begin{aligned} & \text{ADDFUNCTION}(\{\text{MIN-DIST}, [\text{?B}, \text{?C}], [\text{BLOCK}, \text{CART}], \text{REAL}\}) \\ & \text{ADDPRECONDITION}(\text{PUSH}, (= (\text{CART-POSITION ?C}) ?P)) \\ & \text{ADDPRECONDITION}(\text{PUSH}, (\text{FORALL ?B} (\text{IS-BLOCK ?B}) \\ & \quad (\text{AND} (< (- (+ ?P ?FORCE) (\text{BLOCK-POSITION ?B})) (\text{MIN-DIST} \\ & \quad \text{?B ?C})) \\ & \quad \quad (< (- (\text{BLOCK-POSITION ?B}) (+ ?P ?FORCE)) (\text{MIN-DIST ?B} \\ & \quad \text{?C})))) \\ & \text{ADDFLUENTGENERATOR}(\text{MIN-DIST}, \text{ALLPERMUTATIONS}([\text{BLOCKS}, \text{CARTS}], \\ & \text{UNIFORMDISTRIBUTION}(0, 1))) \end{aligned}$$

4.3.5 Interactions

Natural language definition: Dynamic property of behaviors or actions that impacts multiple entities.

Clarifications: Can be spatial, temporal, or other. Can include the point-of-view agent and other entities.

Formal definition: A transformation sequence exhibits interaction novelty if a fluent is added to or changed in the effects of an action. The fluent must name a relevant function that involves 2 or more entities (here, entities are specifically objects of types AGENT, OBJECT, or their subtypes).

$hasInteractionNovelty(t, d) =$
 $\exists f, fname, aname, i: 0 \leq i < |t|$
 $\wedge ((transformation-type(t_i) \in \{ADDACTIONEFFECT, REMOVEACTIONEFFECT\}$
 $\wedge effect-name(transformation-argument(t_i, EFFECT)) = fname)$
 $\wedge \exists a \in domain-actions(d):$
 $aname = action-name(a)$
 $\wedge transformation-argument(t_i, ACTIONNAME) = aname$
 $\wedge function-name(f) = fname$
 $\wedge |entities-in-function(f)| > 1$
 $\wedge d_T = applySequence(t, d)$
 $\wedge (\exists a_T \in domain-actions(d_T) :$
 $action-name(a_T) = aname$
 $\wedge \exists e \in action-effects(a_T) : effect-name(e) = fname)$

Example: A new action allows an agent to move to a nearby cart; this affects the binary function CONTROLS, which becomes dynamic and affects the relationship between agent and cart.

```
ADDACTION(SWITCH-CARTS, [?C1, ?C2], [CART, CART])
ADDPRECONDITION(SWITCH-CARTS, (= (CONTROLS ?AG ?C1) TRUE))
ADDPRECONDITION(SWITCH-CARTS, (= (CONTROLS ?AG ?C2) FALSE))
ADDPRECONDITION(SWITCH-CARTS, (= (CART-POSITION ?C1) ?P1))
ADDPRECONDITION(SWITCH-CARTS, (= (CART-POSITION ?C2) ?P2))
ADDPRECONDITION(SWITCH-CARTS, (< (- ?P2 ?P1) 1))
ADDPRECONDITION(SWITCH-CARTS, (< (- ?P1 ?P2) 1))
ADDACTIONEFFECT(SWITCH-CARTS, (= (CONTROLS ?AG ?C2) TRUE))
ADDACTIONEFFECT(SWITCH-CARTS, (= (CONTROLS ?AG ?C1) FALSE))
```

4.3.6 Environment

Natural language definition: A change in an element of an open-world space that may impact the entire task space and is independent of a specific entity.

Clarifications: Environment novelties may impact the entire task space identically (e.g., changing the temperature from a static value to a new static value) or may impact regions of the open world domain differentially (e.g., wind may be present in only part of a space and the intensity may ebb and flow over time).

Formal definition: An environment novelty is a transformation sequence in which environmental functions, processes, or events are modified. Environmental functions are those functions outside of agent control, meaning they are not affected by functions. Environmental processes and events

are those that are conditioned on or triggered by only environmental functions and position-valued functions.

$$\begin{aligned}
 &hasEnvironmentNovelty(\mathbf{t}, d) = \\
 &(\exists i, d_T: \\
 &\quad 0 \leq i < |\mathbf{t}| \\
 &\quad \wedge d_T = applySequence(\mathbf{t}, d) \\
 &\quad \wedge (transformation-type(\mathbf{t}_i) = ADDFLUENTGENERATOR \\
 &\quad \quad \wedge \exists f, fname: \\
 &\quad \quad \quad isEnvironmentalFunction(f, d_T) \\
 &\quad \quad \quad \wedge fname = function-name(f) \\
 &\quad \quad \quad \wedge transformation-argument(\mathbf{t}_i, NAME) = fname) \\
 &\quad \vee (transformation-type(\mathbf{t}_i) \in \{ADDTRIGGER, REMOVETRIGGER, ADDEVENTEFFECT, \\
 &\quad \quad REMOVEEVENTEFFECT, CHANGEPROBABILITY, CHANGEFREQUENCY\} \\
 &\quad \wedge \exists ev \in domain-events(d_T): \\
 &\quad \quad event-name(ev) = transformation-argument(\mathbf{t}_i, EVENTNAME)) \\
 &\quad \quad \wedge isEnvironmentalEvent(ev, d_T)) \\
 &\quad \vee (transformation-type(\mathbf{t}_i) \in \{ADDPROCESSCONDITION, REMOVEPROCESSCONDITION, \\
 &\quad \quad \quad ADDPROCESSCHANGE, REMOVEPROCESSCHANGE\} \\
 &\quad \quad \wedge \exists p \in domain-processes(d_T): \\
 &\quad \quad \quad process-name(p) = transformation-argument(\mathbf{t}_i, PROCESSNAME)) \\
 &\quad \quad \quad \wedge isEnvironmentalProcess(p, d_T))
 \end{aligned}$$

Required definitions:

$$\begin{aligned}
 &isEnvironmentalFunction(f, d) = \\
 &\quad f \in domain-functions(d) \\
 &\quad \wedge \neg(\exists a, e: a \in domain-actions(d) \wedge e \in action-effects(a) \wedge effect-name(e) = function-name(f)) \\
 &\quad \wedge |entities-in-function(f)| = 0
 \end{aligned}$$

$$\begin{aligned}
 &isEnvironmentalProcess(p, d) = \\
 &\quad \exists cnd \in process-preconditions(ev), f \in domain-functions(d): \\
 &\quad \quad function-in-condition(function-name(f), cnd) \wedge isEnvironmentalFunction(d, f) \\
 &\quad \forall cnd \in process-preconditions(p), f \in domain-functions(d): \\
 &\quad \quad \neg function-in-condition(function-name(f), cnd) \\
 &\quad \quad \vee isEnvironmentalFunction(d, f) \vee function-value-type(f) = POSITION
 \end{aligned}$$

$$\begin{aligned}
 &isEnvironmentalEvent(ev, d) = \\
 &\quad \exists cnd \in event-triggers(ev), f \in domain-functions(d): \\
 &\quad \quad function-in-condition(function-name(f), cnd) \wedge isEnvironmentalFunction(d, f) \\
 &\quad \forall cnd \in event-triggers(ev), f \in domain-functions(d): \\
 &\quad \quad \neg function-in-condition(function-name(f), cnd) \\
 &\quad \quad \vee isEnvironmentalFunction(d, f) \vee function-value-type(f) = POSITION
 \end{aligned}$$

Example: A transformation introduces a “gravitational jump” that distorts pole angles every 20 time units. This is modeled by an environmental event, JUMP-PULLS.

```

ADDFUNCTION(JUMP-TIME, [], [])
ADDPROCESS(JUMP-TICKS, [], [])

```

```
ADDPROCESSCHANGE(JUMP-TICKS, (INCREASE (JUMP-TIME) (* DT 1)))  
ADDEVENT(JUMP-PULLS, [?C], [CART])  
ADDTRIGGER(JUMP-PULLS, (> (JUMP-TIME) 20))  
ADDEVENTEFFECT(JUMP-PULLS, (INCREASE (POLE-ANGLE ?C) 5))  
ADDEVENTEFFECT(JUMP-PULLS, (SET (JUMP-TIME) 0))  
ADDFLUENTGENERATOR(JUMP-TIME, ALLPERMUTATIONS([], UNIFORMRANDOM(0, 20)))
```

4.3.7 Goals

Natural language definition: The purpose of a behavior by an agent in the environment.

Clarifications: Goal novelties resulting from environmental transformations are primarily considered to be goal changes of external agents in the environment that are not the point-of-view agent. Some domains may include teammates where the task goal is dynamic and communicated to the point-of-view agent as a directive or change in observable reward. In such domains, changes in the goals of the team or point-of-view agent communicated by external team members can be considered in-scope, but not required for point-of-view.

Formal definition: Goal novelties are those that include a T-Transformation changing how the performance function of a scenario generator is calculated.

hasGoalNovelty(*t*, *d*) $\equiv$

$\exists t \in \mathbf{t}: \text{transformation-type}(t) = \text{REPLACEPERFORMANCECALCULATION}$

Example: An environment transformation introduces a new performance calculation that rewards an agent lasting without winning or losing.

```
ADDFUNCTION(CLOCK-TIME, [], [])  
ADDPROCESS(CLOCK-TICKS, [], [])  
ADDPROCESSCHANGE(CLOCK-TICKS, (INCREASE (CLOCK-TIME) (* DT 1)))  
REPLACEPERFORMANCECALCULATION((CLOCK-TIME))  
ADDDEFAULT(CLOCK-TIME, 0)
```

4.3.8 Events

Natural language definition: A state change or series of state changes that are not the result of volitional action by an external agent or the point-of-view agent.

Clarifications: Events include state changes with specific preconditions.

Formal definition: An event novelty is a transformation sequence that affects T-SAL events that have both environmental and non-environmental triggers. At least one trigger of a modified event must be based on an environment function and at least one must be directly affected by an action. Event novelty sequences include those that modify the distribution of an environmental function that triggers such an event.

hasEventNovelty(*t*, *d*) $\equiv$

$\exists ev, d_T, i:$

$ev \in \text{domain-events}(d_T)$

$$\begin{aligned} & \wedge d_T = \text{applySequence}(t, d) \\ & \wedge \neg \text{isEnvironmentalEvent}(ev, d_T) \\ & \wedge 0 \leq i < |t| \\ & \wedge ((\text{transformation-type}(t_i) \in \{\text{ADDTRIGGER}, \text{REMOVETRIGGER}, \text{ADDEVENTEFFECT}, \\ & \text{REMOVEEVENTEFFECT}, \text{CHANGEPROBABILITY}, \text{CHANGEFREQUENCY}\} \\ & \quad \wedge \text{event-name}(ev) = \text{transformation-argument}(t_i, \text{EVENTNAME})) \\ & \vee (\text{transformation-type}(t_i) = \text{ADDFLUENTGENERATOR} \\ & \quad \wedge \exists f: \text{isEnvironmentalFunction}(f, d_T) \\ & \quad \quad \wedge \text{transformation-type}(t_i, \text{NAME}) = \text{function-name}(f) \\ & \quad \quad \wedge \exists \text{cnd} \in \text{event-triggers}(ev): \text{function-in-condition}(\text{function-name}(f), \text{cnd})) \end{aligned}$$

Example: An environment transformation introduces a “gravitational anomaly” that pulls nearby carts and blocks based on their position, if they are moving. This is modelled by a non-environmental event, GRAVITY-PULLS, with a varying position trigger, an environmental function trigger, and a third trigger (velocity) under the agent’s control.

```
ADDFUNCTION(GRAVITY-LOCATION)
ADDPROCESS(GRAVITY-WAVE-MOVES, [], [])
ADDPROCESSCHANGE(GRAVITY-WAVE-MOVES, (INCREASE (GRAVITY-LOCATION) (* DT 1)))
ADDEVENT(GRAVITY-WAVE-RESETS, [], [])
ADDTRIGGER(GRAVITY-WAVE-RESETS, (>= (GRAVITY-LOCATION) 20))
ADDEVENTEFFECT(GRAVITY-WAVE-RESETS, (SET (GRAVITY-LOCATION) -20))
ADDEVENT(GRAVITY-PULLS, [?C], [CART])
CHANGEVENTFREQUENCY(GRAVITY-PULLS, 4)
ADDTRIGGER(GRAVITY-PULLS, (> (CART-VELOCITY ?C) .01)))
ADDTRIGGER(GRAVITY-PULLS, (= (CART-POSITION ?C) ?P)))
ADDTRIGGER(GRAVITY-PULLS, (= (GRAVITY-LOCATION) ?WAVE-POS)))
ADDEVENTEFFECT(GRAVITY-PULLS, (INCREASE (CART-VELOCITY ?C) (* DT (LOG-DISTANCE
?P ?WAVE-POS))))
ADDFLUENTGENERATOR(GRAVITY-LOCATION, ALLPERMUTATIONS([], UNIFORMRANDOM(-
20, 20)))
```

5. Publications Developed

5.1 An Environment Transformation-based Framework for Comparison of Open-World Learning Agents

To compare the ability of agents to learn in open worlds, we need a framework with clear definitions of open world environments and how they can vary. This paper provides such a framework, proposing clear scientific definitions for open world environments, transformations of those environments, and characterizations of those transformations. We also provide a description of the minimum necessary conditions under which learning can be expected to occur, and a discussion of how the framework can be used. The Publication can be found in Appendix A.

5.2 Transforming Environments to Evaluate Agent Adaptation

There is considerable recent interest in agents that can achieve goals and maximize performance in transformed environments. The ability of an agent to maintain performance after an environment transformation is sometimes referred to as novelty robustness. However, current test beds for open-world novelty require domain-specific processes to modify environment dynamics to achieve transformations that both require and permit agent adaptation; these processes are time consuming and biased. We introduce a novel, domain-independent environment transformation generator that randomly selects meaningful transformations meeting these requirements. We claim that this generator produces transformations that are learnable: i.e., after such a transformation occurs to its environment, a suitable learning agent can improve its performance online in the environment without human intervention. First, the generator creates permutations of an existing environment model, then it filters them to find those transformations that are relevant to, noticeable to, and controllable by the agent. We report the following contributions: a domain-independent algorithm and implementation for automatically generating environment transformations, an analysis of the space of possible transformations, and experimental evidence that the transformations selected by our generator are learnable. The Publication can be found in Appendix B.

5.3 A Framework for Characterizing Novel Environment Transformations in General Environments

To be robust to surprising developments, an intelligent agent must be able to respond to many different types of unexpected change in the world. To date, there are no general frameworks for defining and characterizing the types of environment changes that are possible. We introduce a formal and theoretical framework for defining and categorizing environment transformations, changes to the world an agent inhabits. We introduce two types of environment transformation: R-transformations which modify environment dynamics and T-transformations which modify the generation process that produces scenarios. We present a new language for describing domains, scenario generators, and transformations, called the Transformation and Simulator Abstraction Language (T-SAL), and a logical formalism that rigorously defines these concepts. Then, we offer the first formal and computational set of tests for eight categories of environment transformations. This domain-independent framework paves the way for describing unambiguous classes of novelty, constrained and domain-independent random generation of environment transformations, replication of environment transformation studies, and fair evaluation of agent robustness. The Publication can be found in Appendix C.

5.4 Human in the Loop Novelty Generation

Recent technological advancements in the field of machine learning and artificial intelligence have increased research effort in the novelty generation process, where thorough testing of machine learning and artificial intelligence models in unexpected situations are crucial in achieving the desired level of logic results. Due to the complexity and abstractness in generating novelties, current novel situations has typically been handmade, making the process time-consuming, inefficient, and inherently human-biased. Current novel situations are developed by

humans who are fully aware and disclosed of the novel situation and have clear approaches in testing those situations with AI. Therefore, training these model logic with human-biased novelties undermines the logic. This paper addresses the current novelty generation process with a human in the loop (HITL) novelty generator that is time efficient and removes human bias. Our proposed HITL novelty generator leverages human domain experts benchmark for good, unanticipated novelties with new automatic novelty generation process that is orders of magnitudes greater than previous novelty generators. The HITL method was used to develop 3 different novelty levels and tested with two domains (Monopoly and VizDoom). Our results shows the HITL method enables users to develop, implement, test, and revise novelties within 5 hours depending on the target domain. The Publication can be found in Appendix D.

6. SAIL-ON TA2 Novelty Analysis

During the option period we provided feedback on SAIL-ON TA2 performers descriptions of novelties. These were provided as feedback to help TA2 decision-making on the SAIL-ON program. Our primary goal was to provide descriptions of the novelty dimensions and estimated difficulties of them, for TA2s to consider; and to provide TA2s with our assessments of how well they meet the novelty hierarchy level as we have formalized it in our theory (see above).

For each novelty given to us by the TA2 (see appendices E-F), we describe the following elements:

- Target Level: The novelty hierarchy level that the TA2 targeted
- Novelty Description: TA2's description of the novelty
- Assumptions: Assumptions we made about the novelty that may or may not be correct.
- Discussion: We use this section to provide context for how we look at the novelty and how we model it.
- Estimated T-SAL Environment Model Elements: Excerpts from a hypothetical T-SAL model of the modified and/or original environment that describe the novelty.
- Estimated T-SAL Transformation Sequence: Formal description of the novelty as a sequence of transformations between initial environment model and scenario generator to modified environment model and scenario generator; as described above.
- Novelty Dimensions: Assignment of novelty characteristics along eight environment transformation dimensions described in our Environment Transformation Framework theory, and explanation of each assignment.
- Hierarchy Level Analysis: An analysis of whether the described novelty fits the definitions of hierarchy levels in the SAIL-ON program, based on our formal theory describing each.

Our assessments of novelty dimension values and hierarchy level analysis are based on the T-SAL language (itself primarily based of PDDL+) and transformations to environments described in that language. As such, the estimated transformation sequence is the formal basis for our determinations, and it's based on the estimated environment model elements required to describe a novelty. In many cases, multiple transformation sequences could create different environment models that would explain the observations generated equally well. When we see this as being possible, we have noted it.

The hierarchy level analysis for each novelty answers three questions:

1. Does the novelty, as described, clearly have the characteristics of the stated level as described in our theory? Might it be described at a lower one?
2. Does the novelty **also** have characteristics from a higher novelty level, which might make it more difficult than intended?
3. Is the novelty as simple as possible? This is related to PM Ted Senator’s request that only a “single” novelty be presented at each point. Our transformation sequences make clear that it is usually not possible to describe a novelty with only a single textual change to a model, so we ask instead whether a simpler transformation sequence can be constructed, by removing one or more existing elements, that still meets the requirements of the target novelty level. There may be circumstances in which meeting this criterion is impractical.

These questions have the following possible responses:

Q	Icon	Response	Meaning
1		<i>Analyzed level matches intended level</i>	Our theory of hierarchy levels treats the level of the novelty as the same one you provided.
		<i>Could be modelled at lower level</i>	We found a model for the novelty as described that matches the provided level in our theory of hierarchy levels, but we also found an alternative model at a lower novelty level. This might confuse some agents.
		<i>Analyzed level does not match intended level</i>	We did not find a model for the novelty that matches your provided level.
2		<i>Higher level novelties not present</i>	The transformation sequence found does not also meet the conditions for a higher level novelty than that stated.
		<i>Higher level novelties are present</i>	The transformation sequence found meets the conditions for a higher level novelty than that stated, in addition to conditions for a lower level.
3		<i>Simpler novelty not possible</i>	There is no way to remove a transformation from the T-SAL transformation sequence that models this novelty and arrive at a sequence from the described level.
		<i>Simpler novelty is possible</i>	One or more T-SAL transformations could be removed from this sequence, and the resulting sequence would still be a legal novelty at the intended level. (That novelty might be a nuisance novelty, though.)

In addition to the icon and response, we describe the rationale for each response, and provide a possible remedy, for “non-green” responses.

7. Software

Under the option period we developed new software that built upon both the theory and previously created software from the base period. Specifically, we extended the TSAL-Parser library to support goal representations, we extended the novelty-generator with refined R-

transformations and add scenario generator capabilities to support T-transformations (which are modifications to a scenario generator). The code described below will be submitted along with this report.

The latest version of the tsal parser can be found at this Github repository:
<https://github.com/Parallax-Advanced-Research/tsal>

The latest version of the novelty generator can be found at this Github repository:
<https://github.com/Parallax-Advanced-Research/noveltygen>

Additionally, we developed a process to automatically filter out relevant novelties in the *data network* PDDL domain in our 2022 ACS publication titled “Transforming Environments to Evaluate Agent Adaptation”. The code for the experiments in this paper is provided as 3 separate zip files (*artue-discoverhistory-ACS2023Experiment.zip*, *novelty-generator-ACS2023Experiment.zip*, and *tsal-interpreter-ACS2023Experiment.zip*).

Our evaluation procedures from this work can be found in the *novelty-generator-ACS2023Experiment.zip* file. The starting point to run the code is *test.py* and the *test_agent()* function runs the planning agent and then collects the execution results of the agent, used to evaluate whether a novelty is **relevant**. If a novelty is not relevant, it is filtered out.

Metrics are computed in lines 224-240 of *test_agent()* in *noveltyGenerator.py* in the *novelty-generator-ACS2023Experiment.zip* file. The primary metric collected is plan cost, which was used to compute relevance values.

Lines 206 - 223 of *test_agent()* call the planner and test agent that tries to solve the novel problems, which is how we measure performance.

When conducting experiments of agents in other domains, such as Monopoly or Vizdoom, we used metrics and measures developed by those TA2 teams, which were consistent with the methodology developed under SAIL-ON.

8. Technical Results

To evaluate the proposed novelty generation process, a series of experiments was conducted in two domains to investigate the limitations and capabilities of the process. The two domains used to evaluate the HITL novelty generation process were (1) Monopoly and (2) VizDoom. The experimental setup for each domain is detailed below. After the work was completed in the Monopoly and Vizdoom domains, the team transitioned to developing the capability to inject novelty and evaluate an agent in a maritime domain. The team is aiming to get a simple system working by the end of the option period in order to have a proof of concept of a real-world application of the concepts developed during this contract.

8.1 Monopoly Domain

The Monopoly domain is a python program that simulates the Monopoly board game that is developed under ISI's GNOME (<https://github.com/mayankkejriwal/GNOME-p3>), an

experimental platform created to evaluate the performance of multi-agent AI systems in the face of novelty. GNOME separates the development of AI game-playing agents from the simulator, allowing for the creation and injection of unanticipated novelty not disclosed to the agent.

8.1.1 Agent Setup

Agents used in this domain can be found in the GNOME-p3 Github repository with the following modified rule set:

1. Players don't receive an extra turn on doubles
2. Players automatically set free from jail after one turn

The experiments consisted of four total agents: one baseline reinforcement learning (RL) agent (Agent 1) and three identical background agents (Agent 2, Agent 3, and Agent 4). The descriptions of these agents can be seen below in Table 5. All three of these background agents follow the same decision-making processes in order to win the game. The background agents achieve victory through proper trades and property management in order to bankrupt the rest of the players. They are not biased against the player agent in the sense that they do not make decisions that are detrimental to ONLY the player agent.

Agent Information	
Name	Description
Agent 1	RL trained agent within monopoly domain
Agent 2	This agent doesn't purchase Red or Orange properties
Agent 3	This agent will not improve properties beyond 2 houses
Agent 4	This agent will always try to buy properties (needs improved mortgage logic to more successfully carry out this strategy)

Table 5 - Summary of developed agents used to verify and validate generated novelties of various levels.

8.1.2 Evaluation Setup

To evaluate the HITL novelty generation process, each of the generated novelties were evaluated by their viability, or impact on the baseline agent. This was done by comparing win rates of agents in pre- and post-novelty tournaments. The average wins and standard deviation of the pre-novelty agent performance is shown below.

Control (Pre-Novelty) Experiment – Agent Win Counts				
Control	Agent 1	Agent 2	Agent 3	Agent 4
Seed = 5	6871	995	1110	1024
Seed = 10	6903	916	1134	1047
Seed = 999	6938	921	1146	995

Table 6- Win counts of all agents in the control (pre-novelty) experiments. This will be used to evaluate our HITL novelty generation process by computing for the change in agent win pre- and post-novelty

To evaluate the HITL novelty generation process, each generated novelties are evaluated by their viability, or impact on the baseline agent by comparing win rates of agents in pre- and post-novelty tournaments. This was used to define the viability of novelties:

- **Low Viability:** $0.66\% < \text{Agent 1} < 4\%$
- **Medium Viability:** $4\% < \text{Agent 1} < 9\%$
- **High Viability:** $\text{Agent 1} > 9\%$

Standard Deviation of Pre-Novelty Agent Performance		
	Agent 1	Default Background Agent
Raw Std Dev	45.66	30.07
% Win Rate Std Dev	0.66	2.96

Table 7 - Standard deviation of pre-novelty agent performance over 20 seeds.

The change in win rate used to assess viability of a novelty could either be an increase or decrease in win rate. In other words, a novelty that alters the baseline agent's performance significantly enough could be considered viable.

For each experiment, we simulated three tournaments with 10,000 instances (games) per tournament. To keep results consistent, we used meta seeds 5, 10, 999 as inputs for the tournaments.

8.1.3 Novelty Level Setup

A total of two novelty levels were generated using the HITL novelty generation process. Each level is defined as:

Level 6: This level aims to observe the agent performance in modified environmental scenarios.

Level 8: This level aims to observe the agent performance under modified event from player actions scenarios.

Experiment Information		
ID	Description	Level
0	Players gain \$1000 when passing GO	6
1	Players lose \$500 when passing GO	
2	Players can only move when dice values are identical	
3	Players gain \$25 at the end of their turn	
4	Player must be in jail to receive a property after trade is complete	8
5	Player must be in jail to receive cash after trade is complete	
6	Property is automatically sold back to the bank after rent is paid for that property	
7	Rent is not paid unless the owner of the space is in jail	
8	The player cannot collect money if they are the list bidder to an auction	

Table 8 - Summary of developed agents used to verify and validate generated novelties of various levels.

8.2 VizDoom Domain

VizDoom is an open-source project used as a research platform for RL using the popular first-person shooter game, Doom (1993), as the environment. This project allows researchers and developers to create and test learning and decision-making algorithms through the modular simulation environment. We developed on top of Washington State University's (WSU) VizDoom Novelty Generator (<https://github.com/holderlb/WSU-SAILON-NG/tree/master/WSU-Portable-Generator>) to inject our custom novelties and conduct experiments on RL agents.

8.2.1 Agent Setup

WSU provided their state-of-the-art (SOTA) player agent for testing, which we used as our baseline agent. While we are unable to go over the specifics of the baseline player agent's RL logic, we are able provide a general overview of its functionality.

The agent has eight actions: forward, backward, left, right, turn left 45 degrees, turn right 45 degrees, shoot, and no action. For each turn, a feature vector of the current state of the simulation environment is sent to the agent; including general information about enemies, items, and the player agent. In response, the agent provides an action to be performed. Performance is defined as the amount of time left in the episode divided by the maximum time for the episode.

Training for the agent is done separately from testing. The user is free to indicate the number of training instances the agent should be trained on. When training is complete, the current model for the agent is saved in memory. We have pre-trained the SOTA agent for 100 pre-novelty instances before using it for novelty testing.

8.2.2 Evaluation Setup

The VizDoom domain consists of a fork of the WSU VizDoom GitHub (<https://github.com/mbercasio94/WSU-SAILON-NG>); with a change of four total enemies and four of each item (health, ammo, trap) spawned for every instance. We conducted one experiment with 1000 instances for pre-novelty, and 1000 instances of each post-novelty. As a control and point of comparison, the SOTA agent was able to win 991 out of 1000 pre-novelty games averaging to a win-rate of 99.1 percent.

8.2.3 Novelty Level Setup

A total of two novelty levels were generated using the HITL novelty generation process. Each level is defined as:

Level 6: This level aims to observe the agent performance in modified environmental scenarios.

Level 8: This level aims to observe the agent performance under modified event from player actions scenarios.

Experiment Information		
ID	Description	Level
0	Player loses 1 health point every 7 turns	6
1	Player loses one ammo clip every 40 turns	
2	Player is turned 90 degrees every 3 turns	
3	After rotating, the player loses 2 health points	8
4	After shifting positions, the player loses 1 health point	
5	After rotating, the player is turned 15 degrees to the left	

Table 9 - Summary of developed agents used to verify and validate generated novelties of various levels.

8.3 Maritime Domain

At this time, the team is aiming to apply the novelty generator process beyond the digital space, towards real-world applications. To do this, the team chose to apply the novelty generation and evaluation concepts used in this project to a maritime domain which aligns with the company's portfolio and will maximize the possibility of transitioning concepts / work generated on this project to follow on work.

The maritime domain work is outlined in figure 2 below. This work is in its initial work progress, and any significant findings have yet to be concluded. The team is planning to utilize the remainder of the option period to continue towards building out this capability. The goal is to have a proof of concept system that automatically generates and implements novelties and evaluates an agent based on its performance. This system will be an example of implementation of the concepts developed in this project.

A unique feature of this work effort is the concept of developing the ability of the system to automatically implement the novelties. This is a feature the team is currently exploring and will include an update in the final version of this report.

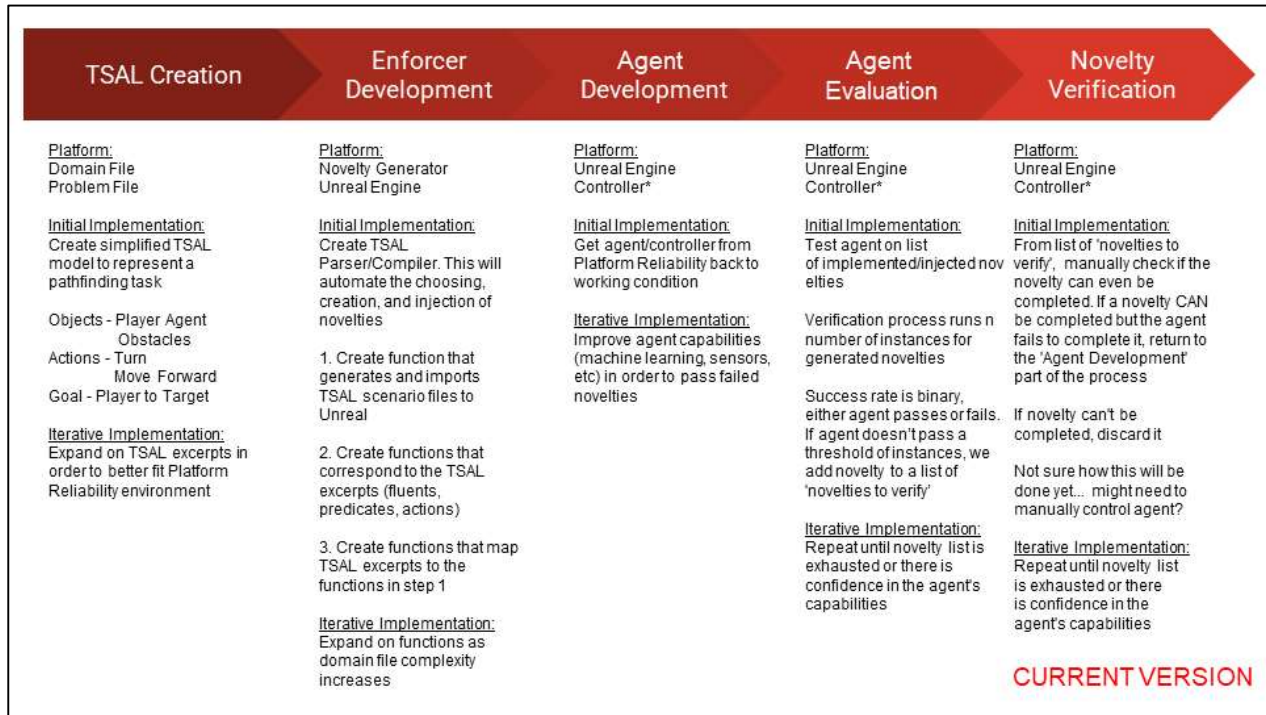


Figure 2 – Maritime Domain Development Flow path

9. Important Findings and Conclusions

During the option period we improved our novelty generation software capabilities and our theoretical formalism of novelty definitions. We implemented a domain-independent filtering capability to determine whether a novelty is relevant – the agent will see reduced performance if it does not address the novelty. We created a human-in-the-loop process where developers can use our novelty generator to create novelties in other simulators, defined outside of TSAL, such as Monopoly and Vizdoom. Overall, we found that our novelty generator is capable of generating infinitely many novelties, and some of these novelties are impactful to agents, as shown in the results in Monopoly and Vizdoom. This work is an important step towards evaluating novelty-robust AI agents by removing human bias from the novelty generation step of verification and validation.

Maritime domain

The goal of this work effort is to get a proof of concept working in a maritime domain. The team will be working to build out this capability throughout the end of the contract. This will provide the team a path to a real-world implementation of the concepts from this project that they can build on for future work / transition efforts. This work utilized the Novelty Generator concepts implemented in the CARLA/Monopoly/Vizdoom domains, and started to apply to it a maritime domain. The development of this work started in the final months of the project, and we are pursuing additional funding for continuing development.

Novelty Generator Conclusions

We introduce the first automated, domain-independent novelty-generator capable of generating infinite environment-based novelties for relational, state-action environments. Our formalism defines learnable novelties as those that are relevant to an agent, noticeable, and controllable. We provide an experimental setup to automatically generate novel environments to obtain new learnable novelties independent of human bias. Using the data-network IPC domain, we evaluated a planning agent on 136 automatically generated novelties, showing that all of these novelties were relevant.

While we believe important steps have been taken with generating relevant novelties, there are paths forward still to explore. While testing relevance, our current methods focused on novelties that affected the transition model. We would like to expand our filtering methods to other novelty types. Besides relevance, we would also like to collect data on the controllability of the generated novelties. Controllability in short is the ability of the agent to make use of the novelty introduced in the domain. We would also like to expand our testing to more domains, which requires a larger representation language. Currently, the language used by the novelty generator can not represent all PDDL domains, and that is a future goal.

Novelty Analysis

During the option period we analyzed 34 Level 6 novelties, 12 Level 7 novelties, and 19 Level 8 novelties for the UTD, WSU, and ISI TA1 teams. We primarily asked 3 questions (see Section 6):

1. Does the novelty, as described, clearly have the characteristics of the stated level as described in our theory? Might it be described at a lower one?
2. Does the novelty *also* have characteristics from a higher novelty level, which might make it more difficult than intended?
3. Is the novelty as simple as possible? This is related to PM Ted Senator's request that only a "single" novelty be presented at each point. Our transformation sequences make clear that it is usually not possible to describe a novelty with only a single textual change to a model, so we ask instead whether a simpler transformation sequence can be constructed, by removing one or more existing elements, that still meets the requirements of the target novelty level. There may be circumstances in which meeting this criterion is impractical.

The results of this analysis can be found in Appendices E-G. Qualitatively, the TA1 teams to which we provided these analyses found them to be helpful, either in categorizing which novelty level their novelty was labelled, and/or making the novelty simpler.

Qualitative Review of Human Experience during Novelty Generation

In this section we go over a HITL's initial thoughts for each step of the novelty generation process. This provides a look at the possible advantages and limitations of our current model in the perspective of a first-time user. Table 9 provides an overview of the time it takes to complete each step for both Monopoly and VizDoom domains.

Step 1 (High Difficulty) Construct TSAL file specific to the target domain:

For those unfamiliar with PDDL or declarative languages in general, it might take several iterations and possible feedback from an expert in order to properly model the domain. Abstraction of a domain must account for certain nuances that might not be so obvious on the initial iteration.

In the case of the Monopoly domain, an extra turn from dice rolls was modelled in the TSAL file even though it was not part of the actual simulation logic, thus resulting in extra turn novelties that deviated too much from the original domain.

Step 2 (Low Difficulty) Run novelty generator using TSAL domain file:

Slight training by an expert or readme is required in order to know which lines of novelty generator code need to be adjusted in order to target specific novelty types.

Step 3 (Low Difficulty) Identify possible novelties from generated files:

As the HITL becomes more familiar with the domain, they should have a better understanding of the novelties that are likely to cause significant impact on agent performance. For example, the HITL may find a pattern where adjusting the preconditions of a specific event has higher likelihood of producing a viable novelty.

Step 4 (Medium Difficulty) Implement the novelty:

Creating novelty injection code depends on the modularity of the domain – how easy is it for a developer to adjust the rules and environment of the simulation space? In addition, difficulty depends on the type of novelty being implemented. For example, novelties that effect agent goals may require further implementation work when compared to a novelty that simply changes an event-based rule of the world. Similar to step 3, as the HITL familiarizes themselves with the codebase, implementation of further novelties should be relatively easier.

Step 5 (Low Difficulty) Test the novelty:

In terms of manual labor, a HITL will need to simply create a batch script in order to run experiments on the implemented novelties. Running experiments with more instances will take longer to complete, but results in a more accurate view of novelty viability. The threshold for the minimum change in performance required for a novelty to be deemed viable will depend on the judgement of the HITL.

Step 6 (Low Difficulty) Revise the novelty:

A HITL will need to simply change parameter(s) in the generated novelty file or novelty generator. As an example, for Experiment 0, Players gain \$1000 when passing GO), the value of \$1000 chosen by the novelty generator may not cause a significant change in agent performance. A user may opt to manually change the value to \$500 which may end up resulting in a more significant change in agent performance. Of course, some novelties will never be viable even with HITL revision. In that case, it is safe to discard the novelty and try a new one.

HITL Novelty Generation Step Description	Avg. Minutes to Complete Step	
	VizDoom	Monopoly
Step 1: Construct TSAL file specific to target domain	Variable	Variable
Step 2: Run novelty generator using TSAL domain file. This process includes editing of the novelty generator parameters to target specific novelty levels. Novelty file generation - target to generate at least 100 files.	5 minutes	5 minutes
Step 3: Identify possible novelties from generated files. Manually parse through generated files for novelties that may prove viable for the target novelty level. Post-processing of certain conditions and effects sometimes required.	30 minutes per novelty	30 minutes per novelty
Step 4: Implement the novelty. Time to implement a novelty depends on the level of that novelty. Initial implementation for a specific novelty level is slow (requires planning of software architecture). Subsequent implementations significantly faster for already created novelty levels.	45 minutes for the first novelty, 20 minutes for subsequent novelties	60 minutes for the first novelty, 20 minutes for subsequent novelties
Step 5: Test the novelty. Run experiment to test for novelty viability. Dependent on computation speed of the simulation machine.	30-60 minutes	1-2 hours
Step 6: Revise the novelty. Manually revise novelty values as needed. May also discard novelty if no changes work.	5 minutes	5 minutes

Table 9 – Summary and time overview for HITL to complete novelty creation steps using novelty generator.

10. Significant Hardware Development

No Hardware was developed during this project

11. Special Comments

Nothing to report under this section

12. Implications for Further Research

In this work, we have set up and formally defined a framework for describing a general class of environments and environment transformations. This is a necessary precursor to general agreement on definitions and measurement of the robustness of artificially intelligent agents. These definitions will be useful to future studies of general agents that take on very difficult situations flexibly and adaptively without human assistance. The “novelty hierarchy” is a first step in exploring possible classes of transformation sequences. Future work will use the formalism described here to define more useful assumptions to guide research. Human design choices about T-SAL environments may have significant impact on how a transformation is described. In some cases, this has been observed to change category membership for the novelty hierarchy described here. This ambiguity is exacerbated when considering agents that have learned their own environment descriptions. To such an agent, the original environment may have a different definition than to a describing human – even if it’s consistent with the same observations. Therefore, a novelty hierarchy categorization might be different than to an observing human.

A known failing of T-SAL-CR is that it does not describe observation of the environment; clearly, the environment is responsible for providing an agent affordances in observation, which can change over time. For example, a robot may gain or lose awareness of its location via a GPS sensor. T-SAL-CR cannot describe what aspects of the state are observable, and therefore the T-SAL transformation language also cannot describe changes in observability. Future work should remedy this problem.

In our HITL research, we introduced a 6-step process by which a human with domain expertise can work alongside our open-source novelty generator to discover, implement, and score AI approaches. Our novelty generator is domain-independent, as demonstrated by our evaluation in two substantially different domains: Monopoly and VizDoom. Any environment that is defined in the Planning Domain Definition Language (PDDL) can be easily modified to work with our novelty generator, since our TSAL language includes most PDDL constructs.

We hope to apply our novelty generator to more environments and develop a more complete testbed environment where agents can be tested directly against the state-transition system without requiring human developer effort to implement novelties in an external simulator. The TSAL language is capable and existing PDDL simulators exist, however at this time a TSAL simulator has not yet been built. A TSAL simulator would enable novelties produced by our method to be used in other frameworks like PDDL-gym: a testbed for reinforcement learning approaches in PDDL domains.

Additionally, leveraging human guidance would be easier with a better user interface. Currently, our system writes novelties to a plain text file that a human can scan line-by-line. An improved user interface should offer features such as syntax highlighting and other visual indicators to aid the human in picking out good novelties, which we leave for future work.

While the results demonstrated our generator's effectiveness in these areas, a significant limitation arises from its restricted scope. Our current model may not generalize well to other domains or interdisciplinary contexts. By limiting the generator's capabilities to only two domains and two novelty levels, we may have inadvertently curtailed its potential to create groundbreaking ideas across a broader range of disciplines. Future research should focus on expanding the generator's reach to encompass multiple domains, thus allowing for a more comprehensive exploration of novel concepts and enhancing its applicability in various fields.

To coincide with the previous statement, the team will be working on a maritime boat automation domain as a proof of concept. With regards to the testing of an automated system in a real-world scenario, the cost of testing could potentially be a limiting factor in the verification and validation of the system. While test cases of the system in a simulated environment may resolve most issues, there still may be edge cases that may not be covered. Expanding on these test cases with the novelty generator could, in theory, cover a lot more potential issues that would incur additional costs in a real-world environment.

13. Other Items of Interest

13.1 Budget Breakdown

Chart 1 shows the Final Cost Breakdown of how contract funds were used to complete the Option period SOW. Table 10 shows a more detailed cost breakdown between PMT / Parallax as well as the overall costs. Final Parallax cost is estimated due to last invoice not being available at the time of the report submittal.

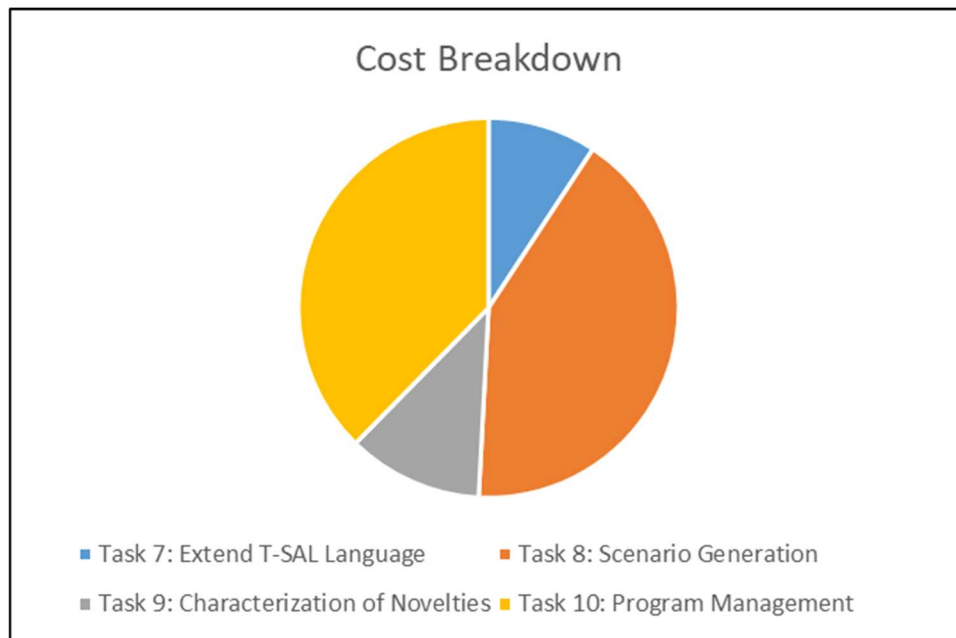


Chart 1: Final Cost Breakdown Chart

Project Objective	PMT		Parallax		Total	
	Amount	%	Amount	%	Amount	%
Task 7: Extend T-SAL Language	\$ -	0%	\$ 42,932	19%	\$ 42,932	8%
Task 8: Scenario Generation	\$ 170,308	60%	\$ 47,317	21%	\$ 217,625	42%
Task 9: Characterization of Novelties	\$ -	0%	\$ 57,001	25%	\$ 57,001	11%
Task 10: Program Management	\$ 115,694	40%	\$ 81,020	35%	\$ 196,714	38%
Labor Total	\$ 286,002		\$ 228,270		\$ 514,272	

Contract Value (Option Period)	\$ 587,639
Labor Total	\$ 514,272
Fee	\$ 32,293
Total Costs	\$ 546,565
Remaining Funds **	\$ 41,074

** Expect some of the remaining funds to be invoiced due to overhead rate increase. This will occur after DCAA audit at end of 2023

Table 10: Final Cost breakdown Tables

14. References

- Bolloor, A., Garimella, K., He, X., Gill, C., Vorobeychik, Y., & Zhang, X. (2019). Attacking Vision-based Perception in End-to-End Autonomous Driving Models.
- Davies, A. (2017, October 31). Inside the Ersatz City Where Waymo Trains Its Self-Driving Cars. Retrieved from Wired.com: <https://www.wired.com/story/google-waymo-self-driving-car-castle-testing/>
- Edelkamp, S., & Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. University of Freiburg.
- Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-208.
- Fox, M., & Long, D. (2003). pddl2.1 : An Extension to pddl for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20, 61-124.
- Gambi, A., Mueller, M., & Fraser, G. (2019). Automatically testing self-driving cars with search-based procedural content generation. *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 318-328). Beijing: ACM.
- Gerevini, A., & Long, D. (2005). Plan Constraints and Preferences in PDDL3 The Language of the Fifth International Planning Competition 1 Motivations and Goals. University of Brescia, Brescia, Italy.
- Helmert, M. (2008). Changes in PDDL 3.1. Retrieved from <http://icapsconference.org/ipc2008/deterministic/PddlExtension.html>
- McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., . . . Wilkins, D. (1998). PDDL | The Planning Domain Definition Language. Yale University, New Haven, CT.
- Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).
- Tesla. (n.d.). Autopilot. Retrieved from Telsa.com: <https://www.tesla.com/autopilotAI>
- Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *Proceedings of the 40th international conference on software engineering* (pp. 303-314). Gothenburg, Sweden: IEEE.
- Uber ATG. (n.d.). Self Driving Cars in Cities. Retrieved from Uber.com: <https://www.uber.com/us/en/atg/cities/>
- Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7), 449-458.
- Balloch, J., Lin, Z., Hussain, M., Srinivas, A., Wright, R., Peng, X., Kim, J., & Riedl, M. (2022). NovGrid: A Flexible Grid World for Evaluating Agent Response to Novelty. *arXiv preprint arXiv:2203.12117*.
- Boult, T. E., Grabowicz, P. A., Prijatelj, D. S., Stern, R., Holder, L., Alspector, J., ... & Scheirer, W. J. (2021). Towards a Unifying Framework for Formal Theories of Novelty. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 17, pp. 15047-15052).

Edelkamp, S., & Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. University of Freiburg.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-208.

Fox, M., & Long, D. (2003). pddl2.1 : An Extension to pddl for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20, 61-124.

Fox, M., & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27, 235-297.

Gamage, C., Pinto, V., Xue, C., Stephenson, M., Zhang, P., & Renz, J. (2021). Novelty Generation Framework for AI Agents in Angry Birds Style Physics Games. *IEEE Conference on Games (IEEE-COG'21)*. Copenhagen, Denmark.

Gerevini, A., & Long, D. (2005). Plan Constraints and Preferences in PDDL3 The Language of the Fifth International Planning Competition 1 Motivations and Goals. Brescia, Italy.

Goel, S., Tatiya, G., Scheutz, M., & Sinapov, J. (2021). Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds. *International Foundation for Autonomous Agents and Multiagent Systems, AAMAS*.

Helmert, M. (2008). Changes in PDDL 3.1. Retrieved from <http://icapsconference.org/ipc2008/deterministic/PddlExtension.html>

Kejriwal, M., & Thomas, S. (2021). A multi-agent simulator for generating novelty in monopoly. *Simulation Modelling Practice and Theory*, 112, 102364.

Kildebeck, E., Holder, L., Kejriwal, L., Renz, J., Robertson, E., & Shrivastava, A. (2022). 2022 Novelty Hierarchy. Unpublished manuscript.

Novelty Working Group. (2023). Working Sessions of the Novelty Working Group [Weekly Meetings]. Virtual Sessions.

Langley, P. (2020). Open-world learning for radically autonomous agents. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 09, pp. 13539-13543).

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., . . . Wilkins, D. (1998). PDDL | The Planning Domain Definition Language. Yale University, New Haven, CT.

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Molineaux, M., & Aha, D. (2014, June). Learning unknown event models. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 28, No. 1).

Molineaux, M. & Dannenhauer, D. (2021). An Environment Transformation-based Framework for Comparison of Open-World Learning Agents. *AAAI Spring Symposium: Designing Artificial Intelligence for Open Worlds*. Virtual.

Muhammad, F., Sarathy, V., Tatiya, G., Goel, S., Gyawali, S. Guaman, M., Sinapov, J., & Scheutz, M. (2021). A Novelty-Centric Agent Architecture for Changing Worlds. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*.

Orwant, J. (2000). EGGG: Automated programming for game generation. *IBM Systems Journal*, vol. 39, no. 3-4, pp. 782-794.

Pell, B. (1992). METAGAME: A new challenge for games and learning. in Heuristic Programming in Artificial Intelligence 3, H. Van den Kerik and L. Allis, Eds. Chichester, U.K.: Ellis Horwood.

Smith, A. M., & Mateas, M. (2010). Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (pp. 273-280). IEEE.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Xue, C., Pinto, V., Zhang, P., Gamage, C., Nikonova, E., & Renz, J. (2022). Science Birds Novelty: an Open-world Learning Test-bed for Physics Domains.

15. Appendices

Appendix A: Parallax Publication: “An Environment Transformation-based Framework for Comparison of Open-World Learning Agents”

Appendix A: Parallax Publication: “An Environment Transformation-based Framework for Comparison of Open-World Learning Agents”

Appendix B: Parallax Publication: “Transforming Environments to Evaluate Agent Adaptation”

Appendix C: Parallax Publication: “A Framework for Characterizing Novel Environment Transformations in General Environments”

Appendix D: PacMar Technologies & Parallax Publication “Human in the Loop Novelty Generation”

Appendix E: Novelty Analysis Results ISI Team

Appendix F: Novelty Analysis Results UTD Team

Appendix G: Novelty Analysis Results WSU Team

An Environment Transformation-based Framework for Comparison of Open-World Learning Agents



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

An Environment Transformation-based Framework for Comparison of Open-World Learning Agents

Matthew Molineaux, Dustin Dannenhauer

4035 Colonel Glenn Hwy, Beavercreek, OH, 45431
first.last@parallaxresearch.org

Abstract

To compare the ability of agents to learn in open worlds, we need a framework with clear definitions of open world environments and how they can vary. This paper provides such a framework, proposing clear scientific definitions for open world environments, transformations of those environments, and characterizations of those transformations. We also provide a description of the minimum necessary conditions under which learning can be expected to occur, and a discussion of how the framework can be used.

Introduction

The challenge of “open world learning”, as we consider it here, requires an agent to be responsive and adapt to changes in its environment’s behavior or observability. Such changes are useful for modelling a variety of difficult unforeseen challenges, such as rare environmental dynamics an agent is unaware of, like tornadoes; discontinuous, unobservable events that modify how actions work, such as a magnetic pole flip; or new sensors or effectors that change an agent’s interface to the environment. While in each of these cases, there is an argument that the environment is not *actually* changing, it is nevertheless expedient to compare agents by changing their environments for the sake of unbiased experimentation.

In this paper, we seek to provide a framework that permits objective measurement and comparison of open-world learners’ efficiency at adapting to changed circumstances. Tools of this framework include general formal definitions of open-world environments, *novelties*, and *novelty regions* that can be applied to any discrete-time environment, and a set of *novelty dimensions*, characteristics that can be used to group and describe novelties for the purposes of classifying open-world learning challenges. We formally define each of these concepts so as to have a solid foundation for discussion of open world challenges.

New research areas require language that differentiates the various challenges they encompass. In action selection problems, we refer to environments as “discrete-time” or

“continuous”, “single-agent” or “multi-agent”, “deterministic” or “stochastic”. It is our hope that the dimensions of novelty introduced here can be used in the same way to describe challenges in this developing field.

Unlike work that seeks to define novelty relative to an agent’s experiences (e.g., Muhammad et al, 2020; Boulton et al, 2021; Gamage et al, 2021), we intentionally avoid any reference to an agent’s knowledge. Such frameworks are useful because they can identify how prepared an agent is for new challenges. However, they cannot usefully describe how environment challenges differ in a way that cuts across differences in knowledge representation. For historical and language reasons, both currently use the term “novelty”, but in different ways.

In the rest of this paper, we review related work on open-world learning, present a novel formal description of multi-agent discrete-time environments, formally define a new set of novelty dimensions that classify environment transformations, provide a set of characteristics necessary for an environment transformation to provide agents an opportunity to learn, and conclude.

Related Work

Boulton et al. (2021) defined a theory of open world novelty where novelty occurs when an agent experiences an environment sufficiently different from its prior experiences. This is fundamentally different from our definition in several ways; first, in our framework, novelty exists independent of any given agent’s experiences or knowledge, for reasons given above. Second, our framework does not define novelty as occurring at a point in time, but rather as encompassing the time-independent difference between two environments. This means that novelty can exist without any changes to the observation or state space. While valuable for considering what experiences could be new to a particular agent, two agents cannot truly be compared on the “same” novelty unless they have identical prior experi-

ences; such comparisons are a key feature of our framework.

Langley’s (2020) work provides a set of broad requirements for a theory of environmental change that can help explain and measure progress in open-world learning. He suggested that such a theory would propose a formalism for environments and transformations on them. Our framework provides both, and so can be consider an example “theory” in this regard that describes environmental changes and provides specific language for characterizing how environment changes vary.

Wiggins’ (2006) framework for creative system in AI describes novelty as a property of a creative output which previously did not exist. While our description of environment transformation novelties makes no restriction on the generating process, it is related: part of the purpose of our framework is to describe constraints on the generation of pairs of environments pre- and post-transformation, where the later one has properties that did not previously exist in the earlier one, so we are broadly consistent with Wiggins’ definition.

Transformation-Based Novelty

Our work depends on a formal definition of novelty and an agent’s interaction with the occurrence of the novelty. The formal definition allows us to describe precisely what is being measured and how the agent is impacted. We now define the formal basis of our definition of novelty from an environment-based perspective.

We describe novelty over a space of discrete-time, turn-taking environments. An environment σ can be described as a tuple $(S, A, O, Ag, Tu, \gamma, \omega)$ where S is the space of possible states, A is the space of possible actions, O is the space of observations, Ag the space of agents, Tu is a turn function $S \rightarrow Ag$ which determines which agent acts in a given state, γ is the probabilistic transition function $S \times A \times S \times Ag \rightarrow \mathbb{R}$, and ω is the probabilistic observation function $S \times O \times Ag \rightarrow \mathbb{R}$. The agent itself is an argument to the observation function, as different agents may have different perceptual interfaces and action capabilities. During environment iteration, in each state s , each actor $ag \in Ag$ receives an observation o according to probability $\omega(s, o, ag)$. Then, one environment-selected agent $Tu(s)$ selects an action a , and the environment transitions to a new state s' with probability $\gamma(s, a, s', Tu(s))$. Other agents cannot take turns, and their probability of transition is 0:

$$\forall s, s' \in S, a \in A, ag \in Ag: \\ \gamma(s, a, s', ag) = 0 \vee Tu(s) = ag.$$

An *environment transformation*, or *novelty*, is a tuple $(\sigma, \sigma^M, \alpha)$ in which σ is an original environment as defined above, $\sigma^M = (S^M, A^M, O^M, Ag^M, Tu^M, \gamma^M, \omega^M)$ is a modified

environment and α is the observing agent. Such a tuple is only a novelty if the transition functions (γ) of the original and modified environment differ or the observation function (ω) differs relative to the observing agent.

We define a “novelty-relevant region” (NR) of the modified state space which encapsulates all states where novelty can be observed or affect the behavior of the agent. As the observation function is agent-relative, the novelty region is as well. It consists of all states for which either the transition function or the observation function has a different outcome in the original and new environments. If two environments are the same, given a particular agent, the novelty region of the new environment will be empty. This can occur if, for example, the novelty only occurs in the observation function relevant to some agents.

$$NR(\alpha, \sigma = (S, A, O, Ag, Tu, \gamma, \omega), \\ \sigma^M = (S^M, A^M, O^M, Ag^M, Tu^M, \gamma^M, \omega^M)) \\ \equiv \{s \in S^M \mid \exists a \in A^M, s' \in S^M: \\ \gamma(s, a, s', Tu(s)) \neq \gamma^M(s, a, s', Tu(s))\} \\ \cup \{s \in S^M \mid \exists o : \omega(s, o, \alpha) \neq \omega^M(s, o, \alpha)\}$$

Outside the novelty region, the two environments will appear and react identically. Naturally, no learning can be expected to take place until an agent enters the novelty region.

In many realistic cases, we expect that particular actions or objects will precipitate the entry of an agent into the novelty region. We expect tracking these to be particularly important to the study of novelty. We will therefore refer to a set of novelty entry transitions, *NET*, that describes where an agent can enter the novelty region.

$$NET(\alpha, \sigma = (S, A, O, Ag, Tu, \gamma, \omega), \sigma^M) \equiv \\ \{(s, a, s') \mid \neg s \in NR(\alpha, \sigma, \sigma^M) \\ \wedge \gamma(s, a, s', Tu(s)) > 0 \\ \wedge s' \in NR(\alpha, \sigma, \sigma^M)\}$$

To permit greater specificity about how actions change in the novelty region, we use factored representations. States are therefore described as assignment functions to a finite vector of discrete state variables $\vec{S} = \{s_0, s_1, \dots, s_n\}$. Each $s \in S$ assigns a value $s(i)$ to every variable s_i in $\vec{S}$. Similarly, O is a space of assignment functions to a vector of discrete and continuous variables $\vec{O} = \{o_0, o_1, \dots, o_n\}$. Each $o \in O$ assigns a value $o(i)$ to every variable $o_i \in \vec{O}$. To describe conditional transition probabilities, we use the notation $P\gamma$ to refer to the probability of changes to individual state variables under a transition function γ . For example, we refer to the probability that the i th state variable takes on a value v following an action a_t taken in state s_t by agent $\alpha \in Ag$ as $P\gamma(s_{t+1}(i) = v \mid a_t, s_t, \alpha)$. The following mathematical equations relate the conditional probabilities of state variable values before and after a transition to the transition function itself:

$$P\gamma(s_{t+1}(i) = v \mid a_t, s_t, \alpha) = \sum_{\{s' \in S \mid s'(i) = v\}} \gamma(s_t, a_t, s', Tu(s_t))$$

$$P\gamma(s_{t+1} \mid a_t, s_t(i) = v) = \sum_{\{s \in S \mid s(i) = v\}} \gamma(s, a_t, s_{t+1}, Tu(s_t))$$

To reason about what is knowable to an agent, we introduce the concepts of state observability and trajectory, which allow us to describe the limits of what an agent can infer based on the information provided by its environment.

We say that a state s is “discernible” when the observation function ω gives an agent α enough information to immediately determine that it is in that state s . Formally, this concept means that every observation o received in state s is not received in other states.

$$\text{discernible}(s, \sigma, \alpha) \equiv \forall s', o: (\omega(s', o, \alpha) > \epsilon \rightarrow s' = s) \vee \omega(s, o, \alpha) < \epsilon$$

In order to reason about what an agent might be able to know, we consider trajectories that describe the totality of an agent’s experiences during an environment interaction. A trajectory τ is a linked list where each element is either the empty trajectory $\emptyset$ or a tuple (a, o, τ') . Here, a is the agent’s selected action if it’s turn is up, or *yield* if it is not; o is the subsequent observation received; τ' is the remaining trajectory. We define an *extension* of a trajectory τ with new action a and observation o as a copy of that trajectory with a new final element $(a, o, \emptyset)$:

$$\text{extension}(\emptyset, a, o) \equiv (a, o, \emptyset)$$

$$\text{extension}(\tau = (a_i, o_i, \tau_i), a, o) \equiv (a_i, o_i, \text{extension}(\tau_i, a, o))$$

The length of a trajectory τ is defined as 0 for $\emptyset$, and 1 + the length of the subtrajectory for all others:

$$\text{length}(\emptyset) \equiv 0$$

$$\text{length}(\tau = (a, o, \tau')) \equiv 1 + \text{length}(\tau')$$

Due to nondeterminism, an agent α with policy π and initial state s may experience any one of multiple possible histories τ of any given length. We define an agent’s decision-making policy $\pi: T \rightarrow A$ as a mapping from the space of trajectories to an action to take at the end of that trajectory. Given a fixed policy π for an agent α in an environment σ , a state sequence s , and past trajectory τ^p , we say that the trajectory τ^f is a *possible future* for agent α when a sequence of states exists, starting with s , such that:

- each action in τ^f is consistent with policy π given the subtrajectory ending prior to it,
- each action in τ^f can lead from the corresponding state in the sequence to the following state, *and*
- each observation in τ^f can be received in the corresponding state in the sequence.

The empty trajectory $\emptyset$ is always a possible future.

$$\text{possible-future}(s = [s_0], \pi, \sigma, \alpha, \tau^p, \emptyset) \equiv \text{True}$$

$$\begin{aligned} \text{possible-future}(s = [s_0 \dots s_n], \pi, \sigma, \alpha, \tau^p, \tau^f) \equiv \\ \sigma = (S, A, O, Ag, Tu, \gamma, \omega) \\ \wedge \tau^f = (a, o, \tau^{f'}) \\ \wedge \pi(\tau^p) = a \\ \wedge a = \text{yield} \vee \exists a': \gamma(s_0, a', s_1, Tu(s_0)) > 0 \\ \wedge a \neq \text{yield} \vee \gamma(s_0, a, s_1, \alpha) > 0 \\ \wedge \omega(s_1, o, \alpha) > 0 \\ \wedge \text{possible-future}([s_1 \dots s_n], \pi, \sigma, \alpha, \\ \text{extension}(\tau^p, a, o)), \tau^{f'}) \end{aligned}$$

Dimensions of Novelty

We have developed a set of dimensions for *classifying* environment-based novelty that are intended to assist in exploration of what makes novelty in the environment challenging for an agent to recognize, characterize, and respond to. We hypothesize that different types of agents will exhibit heterogeneous capabilities in robustness to different dimensions. We also expect dimension values to be correlated with the performance of the agent with respect to different novel environments.

This section is our major contribution. In each of the following eight sections, we give a description of a single dimension along which novelty can vary. This is followed by a bulleted list of possible values for each dimension, each of which is first described informally, then formally. In the same way that terms such as “deterministic”, “discrete” and “multi-agent” are used to describe and differentiate environments for the purposes of action selection, we hope that this framework will serve as a point of reference for discussion of the environment transformations a learning algorithm supports.

In all formal definitions presented in this section, novelty dimension values are presented relative to a novelty $(\sigma, \sigma^M, \alpha)$. Unless otherwise stated, quantified variables with the following base letters are in spaces as follows: $a \in A \cup A^M$, $s \in S \cup S^M$, $o \in O \cup O^M$, $ag \in Ag \cup Ag^M$. Variables with base τ and π are trajectories and policies respectively.

Novel External Agents

In some cases, an agent’s recognition or characterization of environment novelty may be confounded by other agents taking actions whose effects can mask and/or masquerade as changes in the environment. With respect to an agent α , there are two possible conditions:

- **Novel External Agents Present:** Unfamiliar observations can be caused by third parties affecting the environment.

Formally, occurs when a third party agent has the capability to affect environment states with its actions, and adopts a policy in the modified environment that no agents took in the original environment.

$$\exists ag \in (Ag^M \setminus \alpha) \\ \wedge \forall ag' \in Ag: \text{policy}(ag) \neq \text{policy}(ag')$$

- **Novel External Agents Not Present:** No third parties are present that can cause confounding sources of unfamiliar observations.

Formally, occurs when all third-party agents either use the same policies used in the original environment, or have no ability to affect the state of the environment.

$$\forall ag \in (Ag^M \setminus \alpha): \\ \exists ag' \in (Ag \setminus \alpha): \text{policy}(ag) = \text{policy}(ag') \\ \vee \forall a, a', s, s': \gamma^M(s, a, s', ag) = \gamma^M(s, a', s', ag)$$

For example, in a self-driving vehicle domain, an experiment could add an agent with a “drunk driving” policy to the modified environment; even though weaving was possible in the original environment, weaving did not occur because no agents had a policy that caused it. We would then say that “novel external agents present” is a novel dimension of the experiment. However, as policies are not considered part of the environment itself, this can also occur independent of an environment transformation, and outside of a novelty region.

Transition Novelty Type

When a dynamic in the world changes that is observable to an agent α , novelty can be revealed in one of multiple ways, often involving a single variable. These are the simplest types of observable novelty change that can occur. In all cases below, we assume that observability of the changing state variable is high, and that’s why this is the way the agent encounters the novelty. Thus, some pair of observations o and o'' occur that are highly indicative of the particular (actual) states s and s'' to that agent. Some agents may be biased to expect novelty to occur in the form of one of these transitions and neglect others.

- **New change:** Novel transitions cause changes to state variables that were left unchanged in the original environment

Formally, the value of some state variable s_i is always left unchanged when taking some particular action a in some particular state s_t in the original environment, but whose value is sometimes changed when taking action a in state s_t in the modified environment.

$$\exists s_t, a, i, \alpha: \\ P\gamma(s_{t+1}(i) = s_t(i) \mid a, s_t, \alpha) = 1 \\ \wedge P\gamma^M(s_{t+1}(i) = s_t(i) \mid a, s_t, \alpha) < 1$$

- **Missing change:** Novel transitions leave state variables unchanged which were modified in the original environment

Formally, the value of some state variable s_i is sometimes changed when taking some particular action a in some particular state s_t in the original environment, but

whose value is always left unchanged when taking action a in state s_t in the modified environment.

$$\exists s_t, a, i, \alpha: P\gamma^M(s_{t+1}(i) = s_t(i) \mid a, s_t, \alpha) = 1 \\ \wedge P\gamma(s_{t+1}(i) = s_t(i) \mid a, s_t, \alpha) < 1$$

- **Replaced Change:** Novel transitions cause different changes to a state variable than in the original environment

Formally, for some state variable s_i , different changes must occur when taking some particular action a in some particular state s_t in the original environment and modified environment. Specifically, for values c and d , s_i must sometimes take the value c after action a in the changed environment, but never d . In the modified environment, s_i must sometimes take the value d after action a , but never c .

$$\exists s_t, a, i, c, d, \alpha: P\gamma(s_{t+1}(i) = c \mid a, s_t, \alpha) > 0 \\ \wedge P\gamma(s_{t+1}(i) = d \mid a, s_t, \alpha) = 0 \\ \wedge P\gamma^M(s_{t+1}(i) = c \mid a, s_t, \alpha) = 0 \\ \wedge P\gamma^M(s_{t+1}(i) = d \mid a, s_t, \alpha) > 0$$

- **Modified change rate:** The probability of existing transitions is modified.

Formally, there is some state variable s_i whose probability of taking on some value c subsequent to some particular transition is different in the original and modified environments.

$$\exists s_t, a, i, c: P\gamma(s_{t+1}(i) = c \mid a, s_t) > 0 \\ \wedge P\gamma^M(s_{t+1}(i) = c \mid a, s_t) > 0 \\ \wedge P\gamma(s_{t+1}(i) = c \mid a, s_t) \neq P\gamma^M(s_{t+1}(i) = c \mid a, s_t)$$

- **Increase or Decrease in Change Size:** Changes to ordinal-valued state variables are larger or smaller in the modified environment.

Formally, there is some state variable s_i whose probability of increasing by a value greater than c is negligible in the original environment but non-negligible in the modified environment, or whose probability of increasing by a value less than c is negligible in the original environment but non-negligible in the modified environment.

$$\exists s_t, a, i, c: \\ P\gamma(s_{t+1}(i) - s_t(i) > c \mid a, s_t, \alpha) < \epsilon \\ \wedge P\gamma^M(s_{t+1}(i) - s_t(i) > c \mid a, s_t, \alpha) > \epsilon \\ \vee P\gamma(s_{t+1}(i) - s_t(i) > c \mid a, s_t, \alpha) > \epsilon \\ \wedge P\gamma^M(s_{t+1}(i) - s_t(i) > c \mid a, s_t, \alpha) < \epsilon$$

Novelty Entry Determinism

This dimension concerns the agent’s transition into the novelty region. Infrequent or agent-caused novelty entry may lower the likelihood that the agent encounters the novelty at all, whereas different agent’s biases may allow them to more easily represent deterministic or nondeterministic transitions.

- **Deterministic:** Entry into the novelty region is deterministic.

$$\forall (s, a, s') \in \text{NET}(\alpha, \sigma, \sigma^M): \gamma(s, a, s', \alpha) = 1$$

- **Frequent Nondeterministic:** Some transitions into the novelty region are non-deterministic, and at least one is high probability.

$$\exists (s, a, s') \in \text{NET}(\alpha, \sigma, \sigma^M): \gamma(s, a, s') \geq 0.5 \wedge \gamma(s, a, s') < 1$$

- **Infrequent Nondeterministic:** All transitions into the novelty region are low probability, which may make the novelty region difficult to discover or characterize.

$$\forall (s, a, s') \in \text{NET}(\alpha, \sigma, \sigma^M): \gamma(s, a, s') < 0.5$$

- **Agent-Caused:** The agent is in control of all entry into the novelty region.

Formally, on the agent's turn, there must be some action that cannot transition to the novelty region; and on other agent's turns, there must be no action that can.

$$\forall s \exists a \forall s': \neg Tu(s) \vee \neg (s, a, s') \in \text{NET}(\alpha, \sigma, \sigma^M) \\ \wedge \forall s, a, s', ag: \neg (s, a, s') \in \text{NET}(ag, \sigma, \sigma^M) \vee ag = \alpha$$

Novelty Obviousness

This dimension concerns the minimum amount of observation and reasoning required to determine that the agent's present environment is not the original environment. This will affect the amount of time and concentration required for an agent to recognize novelty.

- **Alien observation:** Observations occur in the modified environment that are not present in the original

Formally, there is some observation o that is in the observation space of the modified environment but not the original environment, *and* there is a non-zero probability that the agent α will receive that observation in some state s .

$$\exists s, o \in O^M: \neg (o \in O) \wedge \omega^M(s, o, \alpha) > 0$$

- **Entry Transition Discernible:** An observation occurs on entry into the novelty region that is not consistent with the original environment.

Formally, for each novelty entry transition (s_t, a, s_{t+1}) , there is a set of corresponding non-novel transitions in the original environment that start in the same state (s_t) and take the same action (a) but lead to a different "original" state (s_{t+1}^O) (one not in the novelty region). Entry observability means two things: First, for every observation the probability of receiving that observation in the original environment must be arbitrarily small in either the novelty entry transition origin state (s_t) or every other. (Thus receiving that observation nearly always means the agent is in that particular state.) Second, for every observation the probability of receiving that observation must be arbitrarily small in either every original state

(s_{t+1}^O) reached by a from s_t or the novelty region entry state (s_{t+1}^M). (Thus it is possible to tell that the transition is not one that exists in the original environment.)

$$\forall o \in (O \cup O^M), (s_t, a, s_{t+1}^M) \in \text{NET}(\alpha, \sigma, \sigma^M): \\ \text{discernible}(s_t, \sigma^M, \alpha) \\ \wedge \forall s_{t+1}^O: \gamma(s_t, a, s_{t+1}^O, \alpha) < \epsilon \\ \vee \forall o: \omega(s_{t+1}^O, o, \alpha) < \epsilon \\ \vee \omega^M(s_{t+1}^M, o, \alpha) < \epsilon$$

- **Novel Transition Discernible:** Some novel transition available to the agent can be observed unambiguously. *Formally*, there is some modified environment transition that starts in the novelty region that cannot happen in the original environment. Both the originating state and the destination state of this transition are unambiguously recognizable, because observations received in these states are not received in other states in the original environment.

$$\exists s_t \in \text{NR}(\alpha, \sigma, \sigma^M), a, s_{t+1}: \gamma^M(s_t, a, s_{t+1}, \alpha) > 0 \\ \wedge \gamma(s_t, a, s_{t+1}, \alpha) = 0 \\ \wedge \text{discernible}(s_t, \sigma, \alpha) \\ \wedge \text{discernible}(s_{t+1}, \sigma, \alpha)$$

- **Novelty Region Inferable:** Given enough time, an agent will obtain enough information to infer that it has experienced some novel transitions and/or observations.

Formally, for every state s in the novelty region and every policy π , all changed environment (σ^M) possible futures (τ^F) of a length greater than n are trajectories that are not possible futures starting in any state s' in the original environment σ , no matter how the state s or s' was reached.

$$\forall s = [s_0 \dots s_m], \pi, \tau^P, \exists n, \forall \tau^F: \\ \text{length}(\tau^F) = m \\ \wedge m > n \\ \wedge s_0 \in \text{NR}(\alpha, \sigma, \sigma^M) \\ \wedge \neg \text{possible-future}(s, \pi, \sigma^M, \alpha, \tau^P, \tau^F) \\ \rightarrow \forall s': \neg \text{possible-future}(s', \pi, \sigma, \alpha, \tau^P, \tau^F)$$

- **Novelty Region Investigable:** Given some policy, an agent can obtain sufficient information to infer after entering the novelty region that the agent had experienced some novel transitions and/or observations.

Formally, there is some traversal s starting in the novelty region of the modified environment σ^M whose trajectory τ^F is not a possible future in the original environment, for any traversal s' .

$$\exists s = [s_0 \dots s_n], \pi, \tau^P, \tau^F: \\ \text{possible-future}(s, \pi, \sigma^M, \alpha, \tau^P, \tau^F) \\ \wedge s_0 \in \text{NR}(\alpha, \sigma, \sigma^M) \\ \wedge \forall s': \neg \text{possible-future}(s', \pi, \sigma, \alpha, \tau^P, \tau^F)$$

- **Statistical Novelty Only:** Some changes may require statistical analysis to verify; all histories possible in the modified environment are also possible in the original.

Formally, any trajectory possible in the modified environment is also possible in the original environment (possibly visiting different states).

$$\begin{aligned} \forall s = [s_0 \dots s_n], \pi, \tau^P, \tau^F: \\ \text{possible-future}(s, \pi, \sigma^M, \alpha, \tau^P, \tau^F) \rightarrow \\ \rightarrow \exists s': \text{possible-future}(s', \pi, \sigma, \alpha, \tau^P, \tau^F) \end{aligned}$$

- **Indistinguishable:** Some changes to environments might never be directly observable to an agent; we generally consider these to be “unfair” or “uninteresting” changes as they afford no learning opportunity.

Formally, for every possible future τ^F , policy π , and state s , there is an equivalent conditional probability of experiencing that future τ^F in the original environment σ and from some state s' in the changed environment σ^M . (This makes the two environments indistinguishable from the agent’s perspective, although the actual states and transitions may have changed.)

$$\begin{aligned} \forall \pi, s, \tau^P, \tau^F \exists s': \\ \Pr(\tau^F | s, \tau^P, \pi, \sigma, \alpha) = \Pr(\tau^F | s', \tau^P, \pi, \sigma^M, \alpha) \end{aligned}$$

Novelty Observation Determinism

This dimension concerns how likely the agent is to receive a novel observation if one is possible. More infrequent observations may be harder to learn as an agent may disregard them as noise. Additionally, many agents may be biased to assume deterministic or nondeterministic observations.

- **Deterministic:** Novel observation mappings are deterministic.

Formally, the probability of receiving any given observation o in a particular state s is the same in both environments, or is exactly 0 or 1 in the modified environment.

$$\begin{aligned} \forall s, o: \omega^M(s, o, \alpha) = \omega(s, o, \alpha) \\ \vee \omega^M(s, o, \alpha) = 0 \vee \omega^M(s, o, \alpha) = 1 \end{aligned}$$

- **Frequent Nondeterministic:** Novel observation mappings are deterministic or happen 50% more or less often in the modified environment than the original.

Formally, the probability of receiving any given observation o in a particular state s is the same in both environments, or is exactly 0 or 1 in the modified environment, or the difference between the probabilities is greater than 0.5.

$$\begin{aligned} \forall s, o: \omega^M(s, o, \alpha) = \omega(s, o, \alpha) \\ \vee \omega^M(s, o, \alpha) = 0 \vee \omega^M(s, o, \alpha) = 1 \\ \vee |\omega^M(s, o, \alpha) - \omega(s, o, \alpha)| > 0.5 \end{aligned}$$

- **Infrequent Nondeterministic:** Some novel observation mappings occur infrequently or with close to the same frequency as in the original environment.

Formally, there is some state s and observation o such that the probability $\omega^M(s, o, \alpha)$ of an agent α receiving

that observation o in that state s in the modified environment is neither the same as in the original environment, nor exactly 0, nor exactly 1; instead, it is within 0.5 of the original environment probability $\omega(s, o, \alpha)$ of receiving the same observation o in the same state s .

$$\exists s, o: |\omega^M(s, o, \alpha) - \omega(s, o, \alpha)| < 0.5$$

Novelty Observation Localization

This dimension concerns the danger implicit in encountering a meaningful change in state before sensing that something is different; in some environments, it is possible for the agent to avoid novelty that degrades its performance, and in others not.

- **Nearby:** Agents encounter novel transitions before observations; that is to say, the environmental novelty will start to affect the agent before the agent would easily notice it.

Formally, any possible trajectory τ in the modified environment σ^M that leads to a novel observation o , must either be also possible in the original environment (with some other state sequence) or be impossible prior to the new observation (i.e., due to a previous novel transition).

$$\begin{aligned} \forall s = [s_0 \dots s_n], \pi, \alpha, \tau, \tau', a, o: \\ \omega(s_n, o, \alpha) = 0 \\ \wedge \text{length}(\tau) = n - 2 \\ \wedge \tau' = \text{extension}(\tau, a, o) \\ \wedge \text{possible-future}(s, \pi, \sigma^M, \alpha, \emptyset, \tau') \\ \wedge \neg \text{possible-future}(s, \pi, \sigma, \alpha, \emptyset, \tau') \\ \rightarrow \neg \text{possible-future}([s_0 \dots s_{n-1}], \pi, \sigma, \alpha, \emptyset, \tau) \end{aligned}$$

- **Distant:** Agents encounter novel observations before novel transitions; they will observe surprising observations before being affected by the environment novelty.

Formally, for any traversal of the modified environment that ends with a novel transition, if an agent’s trajectory τ' over that traversal is not possible in the original environment, the subtrajectory τ ending just before the final transition is also not possible in the original environment.

$$\begin{aligned} \forall s = [s_0 \dots s_n], \pi, \alpha, \tau, \tau', a, o: \\ \gamma(s_{n-1}, a, s_n, Tu(s_{n-1})) = 0 \\ \wedge \text{length}(\tau) = n - 1 \\ \wedge \tau' = \text{extension}(\tau, a, o) \\ \wedge \text{possible-future}(s, \pi, \sigma^M, \alpha, \emptyset, \tau') \\ \wedge \neg \text{possible-future}(s, \pi, \sigma, \alpha, \emptyset, \tau') \\ \rightarrow \neg \text{possible-future}([s_0 \dots s_{n-1}], \pi, \sigma, \alpha, \emptyset, \tau) \end{aligned}$$

- **Mixed:** Sometimes novel observations are encountered first, sometimes novel transitions. We define mixed environments as those that are neither nearby nor distant.

Novel State Observation Activity

This dimension concerns whether effort is required for an agent to observe what is different about the environment.

Some agents may fail to put in effort necessary to find information that will prove beneficial later.

- **Passive:** The agent's observation of effects does not require active effort. Extra action is not required to distinguish novel states.

Formally, for any two states in the modified environment, s and s' , that cannot be distinguished by observation: every action a that leads from s to s'' with non-zero probability has an equivalent transition with identical probability from s' to s'' , and s'' and s''' are indistinguishable by observation.

$$\begin{aligned} \forall s \in S^M / S, s', a, s'': \\ (\forall o \in O^M: \omega^M(s, o, \alpha) = \omega^M(s', o, \alpha) \\ \wedge \gamma^M(s, a, s'', \alpha) > 0) \\ \rightarrow \exists s''': (\gamma^M(s, a, s'', \alpha) = \gamma^M(s', a, s''', \alpha) \\ \wedge \forall o': \omega^M(s'', o', \alpha) = \omega^M(s''', o', \alpha)) \end{aligned}$$

- **Active:** The agent must make a special effort to detect novel transitions. Some novel states may be indistinguishable from other states without specific action. *Formally*, for some pair of transitions s, a, s'' , and s', a, s''' in the modified environment, at least one of which (s) is in the modified environment state space but not the original environment state space, the two origination states (s, s') cannot be distinguished, but the two end states (s'', s''') can.

$$\begin{aligned} \exists s \in S^M / S, s', s'', s''', a: \\ \forall o: \omega^M(s, o, \alpha) = \omega^M(s', o, \alpha) \\ \wedge \gamma^M(s, a, s'', \alpha) > \epsilon \wedge \gamma^M(s', a, s''', \alpha) > \epsilon \\ \wedge \exists o': \omega^M(s'', o', \alpha) \neq \omega^M(s''', o', \alpha) \end{aligned}$$

- **Resource Expenditure Required:** There are environment actions designed for observation that are needed to distinguish novel states, such as a limited-use Geiger counter that detects novel sources of radiation. *Formally*, for any two states in the modified environment, s and s' , that cannot be distinguished by observation, there are two types of outgoing actions: observation actions and non-observation actions. Non-observation actions that lead from a state s to s'' with non-zero probability have equivalent transitions with identical probability from s' to s'' , and s'' and s''' are indistinguishable by observation. Observation actions allow them to be distinguished, and strictly decrease reachability. Here, we define the set of states reachable from a set of states s :

$$\begin{aligned} adjacent(s, s', \sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \equiv \\ \exists a \in A, s \in s: \gamma(s, a, s') > 0 \\ reach(s, \sigma) \equiv \\ s \cup reach(s \cup \{s' \in S \setminus s \mid adjacent(s, s', \sigma)\}) \\ \forall s \in S^M / S, s', a, s'': \\ (\forall o \in O^M: \omega^M(s, o, \alpha) = \omega^M(s', o, \alpha) \\ \wedge \gamma^M(s, a, s'', \alpha) > 0) \\ \rightarrow (\exists s''': \gamma^M(s, a, s'', \alpha) = \gamma^M(s', a, s''', \alpha)) \end{aligned}$$

$$\begin{aligned} \wedge \forall o': \omega^M(s'', o', \alpha) = \omega^M(s''', o', \alpha) \\ \vee reach(\{s''\}, \sigma) \subset reach(\{s'\}, \sigma) \\ \wedge \forall s''': \gamma^M(s', a, s''', \alpha) = 0 \\ \vee reach(\{s'''\}, \sigma) \subset reach(\{s'\}, \sigma) \end{aligned}$$

Novelty Frequency

This dimension concerns how frequently novel changes can occur. Changes that are more frequent than an agent can see, or continuous, may be significantly more difficult for some agents to track and respond to.

- **Less frequent than Observation:** Commonly, observations occur frequently enough to observe important events, so a novel change will occur at most once between observations.
- **More frequent than Observation:** Some novelty sources may have a periodicity faster than an agent's observations, such that multiple changes may occur between one observation and the next.
- **Continuous:** Novelty sources may cause change that is continuous over time, such as the water level in a reservoir.

We leave the formalization of this dimension to be addressed within a continuous time framework.

Fairness

It is easy to construct novelties that are clearly distinct but very difficult to learn about. For example, states may be different between two environments only in some subtle way that is not visible to the agent's sensors. Some unobservable novel event might cause an agent's immediate destruction, allowing no response and providing no useful information the agent could use to avoid that event in the future. Novel transitions might occur only in a distant region of the state space that the agent never encounters in practical situations. We consider some possible conditions that may be necessary to ensure fairness in novelty experimentation.

- **Relevant:** Novelty will affect the agent's performance directly
- **Noticeable:** The agent will have the opportunity to witness something different about the new environment.
- **Controllable:** The agent will have sufficient power to do something about the novelty

To formally define these qualities, we require the following definitions and assumptions. We assume that an agent is motivated by some environment-independent performance measure that is known or observable to it. This function, *Performance*: $T \times S \rightarrow \mathbb{R}$ is some mapping from the space of possible trajectories and actual states traversed to the real numbers; this function is calculated over the full sequence of states and actions in the agent's past trajectory. The space of final states of an environment σ , $S_F(\sigma) \subset S$ is

defined as the set of states which have no outgoing transitions:

$$S_F(\sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \equiv \{s \in S \mid \forall a \in A, s' \in S : \gamma(s, a, s') = 0\}$$

We then define the expected performance of a policy in an environment based on a past trajectory τ^P , state history s , and policy π as the performance of the past trajectory in any final state, or otherwise as the average expected performance of the policy over each possible next state and observation as given by recursion over the next state and updated trajectory.

$$\begin{aligned} EPP(\tau^P, \pi, s = (s_0 \dots s_n \in S_F(\sigma)), \alpha, \sigma) \\ &\equiv Performance(\tau^P, s) \\ EPP(\tau^P, \pi, s = (s_0 \dots s_n \notin S_F(\sigma)), \alpha, \\ &\sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \\ &\equiv \sum_{s' \in S, o \in O} \begin{pmatrix} \gamma(s_n, \pi(\tau^P), s', \alpha) \\ * \omega(s', o, \alpha) \\ * EPP(extension(\tau^P, \pi(\tau^P), o), \\ \pi, concat(s, s'), \alpha, \sigma) \end{pmatrix} \end{aligned}$$

Formally, the novelty of a modified environment σ^M relative to an original environment σ is relevant to an agent α in some starting state s_0 iff there is some policy π that produces different expected performance in σ and σ^M .

$$\begin{aligned} Relevant(\alpha, \sigma, \sigma^M, s_0) \equiv \\ \exists \pi: EPP(\emptyset, \pi, s_0, \alpha, \sigma) \neq EPP(\emptyset, \pi, s_0, \alpha, \sigma^M) \end{aligned}$$

The novelty of a modified environment σ^M relative to an original environment σ is noticeable iff it gains enough experiences of σ^M during training to reject the hypothesis that those observations came from σ . We omit the full formal definition of noticeability to conserve space.

The novelty of an environment is considered controllable by an agent α if there is some policy π^* that performs better for α in the modified environment σ^M than the optimal policy π^* for the original environment σ .

$$\begin{aligned} Controllable(\alpha, \sigma, \sigma') \equiv \\ \exists \pi^* \forall \pi, s \in S: \\ EPP(\emptyset, \pi, s, \alpha, \sigma) \leq EPP(\emptyset, \pi^*, s, \alpha, \sigma) \\ \wedge \exists \pi' \forall s \in S^M: \\ EPP(\emptyset, \pi^*, s, \alpha, \sigma') \leq EPP(\emptyset, \pi', s, \alpha, \sigma') \end{aligned}$$

While these constructions ensure fairness, they also may make some problems easier than desired; future work will continue investigations into means of ensuring the fairness conditions without simplifying the target problem.

Discussion

New work on learning in open-world environments has many varied challenges. We’ve presented a new formal

basis for describing and differentiating those challenges, that we hope will be useful in the comparison of new agent algorithms that are suited to certain types of challenges.

We have also considered the types of problems that are practically solvable. The scope of possible environment transformations is immense, and these basic “fairness” conditions start to consider what the limits of possible performance may be.

We recognize several areas of potential improvement to this framework. First, the discrete-time, turn-taking formulation of the environments is awkward, compromising the simplicity of a Markov decision process-like representation to acknowledge the presence of multiple agents. It also prevents us from formally defining novelty frequency. A future version of this framework will incorporate continuous-time change.

Second, while classifications of problems are useful, we have not yet defined any numeric measurements that can be made, which could strengthen our ability to compare.

Third, we expect refinements to this framework in the form of added dimensions, revised novelty dimension value definitions, and added values as errors are detected and affordances found. We have attempted to find orthogonal dimensions, but relationships between them are likely and should be identified.

While we believe it is important to consider environments separate from agent knowledge, this does present certain difficulties. When comparing agent’s learning speed, it will be important to consider that some agents have advantages due to “accidental” model compatibilities. For example, if an environment transformation requires vehicles to drive on the left rather than the right, an agent whose model predicts negative consequences for “driving toward oncoming vehicles” will perform much better than one whose model predicts negative consequences for “driving in a left lane”, even though they may have performed identically in their original environment. Considering how to measure or prevent such unfair advantages will be important to future experimentation.

References

- Boult, T. E., Grabowicz, P. A., Prijatelj, D. S., Stern, R., Holder, L., Alspector, J., ... & Scheirer, W. J. (2021). Towards a Unifying Framework for Formal Theories of Novelty. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 17, pp. 15047-15052).
- Gamage, C., Pinto, V., Xue, C., Stephenson, M., Zhang, P., & Renz, J. (2021). Novelty Generation Framework for AI Agents in Angry Birds Style Physics Games. *IEEE Conference on Games (IEEE-GameCom’21)*. Copenhagen, Denmark.
- Langley, P. (2020). Open-world learning for radically autonomous agents. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 09, pp. 13539-13543).

Muhammad, F., Sarathy, V., Tatiya, G., Goel, S., Gyawali, S. Guaman, M., Sinapov, J., & Scheutz, M. (2021). A Novelty-Centric Agent Architecture for Changing Worlds. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*.

Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7), 449-458.

Transforming Environments to Evaluate Agent Adaptation



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

Transforming Environments to Evaluate Agent Adaptation

Abstract

There is considerable recent interest in agents that can achieve goals and maximize performance in transformed environments. The ability of an agent to maintain performance after an environment transformation is sometimes referred to as *novelty robustness*. However, current test beds for open-world novelty require domain-specific processes to modify environment dynamics to achieve transformations that both require and permit agent adaptation; these processes are time consuming and biased. We introduce a novel, domain-independent environment transformation generator that randomly selects meaningful transformations meeting these requirements. We claim that this generator produces transformations that are *learnable*: i.e., after such a transformation occurs to its environment, a suitable learning agent can improve its performance online in the environment without human intervention. First, the generator creates permutations of an existing environment model, then it filters them to find those transformations that are relevant to, noticeable to, and controllable by the agent. We report the following contributions: a domain-independent algorithm and implementation for automatically generating environment transformations, an analysis of the space of possible transformations, and experimental evidence that the transformations selected by our generator are learnable.

1. Introduction

AI systems fail during deployment because real-world environments are unpredictable. The problem of responding to “unknown unknowns” that may crop up in an agent’s environment is large; we refer to this as open-world novelty, and consider evaluating agent’s response to it. Specifically, this type of novelty concerns changes (or apparent changes) to the structure or dynamics of the environment an agent inhabits, referred to as an *environment transformation*. Recent work emphasizes robustness and adaptation to such transformations (cite papers here). From an evaluation perspective, existing studies create ad hoc examples of environment transformations that are human-designed, and therefore biased. It is not clear that robustness to human-generated transformations indicates the ability to respond to unknown unknowns. To better evaluate these capabilities, we introduce the first automated *novelty generator* that randomly generates environment transformations to study open-world novelty.

We claim that novelty can be found by sampling a space that is too large to fully enumerate. Our generator searches through a space of environment transformations for examples that are *learnable*; that is to say, they meet certain basic criteria that allow and motivate an agent to learn: relevance, noticeability, and controllability. Learnable environment transformations challenge an agent to respond differently than their original environments. Our approach to novelty generation is consistent with Wiggins’ (2006) Creative Systems Framework (CSF). To our knowledge, our novelty generator is the first instantiation of CSF for relational state-action environment creation.

Note: other work sometimes uses the term "novelty" to refer specifically to novel agent experiences (e.g., Muhammad et al, 2020; Boulton et al, 2021; Gamage et al, 2021); this perspective is valid, but we intentionally avoid any reference to an agent's knowledge in this work so as to avoid experimentation issues across agents with different knowledge representations.

In this paper, we describe a space of transformations over PDDL environments, and a novelty generator that samples that space to find challenging problems in open-world novelty. To demonstrate the success of this algorithm, we present experimental results describing the change in performance of a replanning agent that attempts to achieve goals in the presence of such transformations. We claim that the novelty generator selects transformations without bias, and that the transformations selected present a challenge to an agent.

2. Defining Environment Transformations

We describe environment transformations over a space of discrete-time, turn-taking environments. An environment σ can be described as a tuple $(S, A, O, Ag, Tu, \gamma, \omega)$ where S is the space of possible states, A is the space of possible actions, O is the space of observations, Ag the space of agents, Tu is a turn function $S \rightarrow Ag$ that determines which agent acts in a given state, γ is the probabilistic transition function $S \times A \times S \times Ag \rightarrow \mathbb{R}$, and ω is the probabilistic observation function $S \times O \times Ag \rightarrow \mathbb{R}$. The agent itself is an argument to the observation and transition functions, as different agents may have different perceptual interfaces and action capabilities. During environment interaction, each actor $\alpha \in Ag$ receives an observation o when it enters a state s according to probability $\omega(s, o, \alpha)$. Then, one environment-selected agent $Tu(s)$ selects an action a , and the environment transitions to a new state s' with probability $\gamma(s, a, s', Tu(s))$. Their probability of transition due to an action from any other agent is 0:

$$\alpha \neq Tu(s) \implies \gamma(s, a, s', \alpha) = 0 \quad (1)$$

An environment transformation, or novelty, is a tuple $(\sigma, \sigma^M, \alpha)$ that describes two environments that produce different experiences for an agent. Here, σ is the original environment, $\sigma^M = (S^M, A^M, O^M, Ag^M, Tu^M, \gamma^M, \omega^M)$ is a modified environment, and α is an agent that can interact with either. To produce different experiences, the environments' transition functions (γ, γ^M) must differ, or their observation functions (ω, ω^M) must sometimes produce different observations for agent α .

3. Transformation Space Description

We formally define and implement 24 transformations on PDDL+ domain descriptions; these transformations are the mechanism by which a new, novel domain description is produced. A summary of these transformations and a lower bound on the size of the space of novelties they produce are given in Table 2. Each transformation yields a number of novel domains; this value is given in the *Number of Instances* column. We argue that since it is unrealistic to expect a human to enumerate all transformations to an environment, the space of new environments created by performing these

Key: Parameters	
m=2, n=2, k=2, j=2, v=100	
Domain	Space of Novelties after 1 Transformation
Data Network	1,570,323

Table 1. Estimate Lower Bound Space of Novel Domains for the Data Network domain

transformations will contain novelty. Applying multiple transformations to an original environment yields a space of novelties that is significantly large, increasing in size with each iteration. While the space of novelties given for most transformations is straightforward, we highlight a few individual transformations here: *addDerivedPredicate* has $2 \cdot \binom{C}{j} \cdot 2 \cdot (j-1)!$ instances because $\binom{C}{j}$ are the conditions chosen for the preconditions, then the first 2 is because each condition can be positive or negated, and then the conditions are arranged in **and** and **or** (hence the second 2) and we will consider all combinations (explaining the !) of up to $j-1$ pairs of conditions; *addNewType* has an optional argument of *typeParent* and if no type parent is given, the number of instances is 1, similar to *addProcess*. In cases where a new predicate or type is added to the domain, the dynamics of the environment do not change; the added construct must be part of a state transition to affect an agent’s experience (i.e. part of an action/event/process’s preconditions or effects).

A lower bound number of new environments N produced by applying all transformations is:

$$\begin{aligned}
 &1 + |F| + |A| + |D| + 2|U| + 3|E| + (4+k)|T| + \\
 &|T \times T| + |U \times C| + |(A+E) \times C^{ex}| + \\
 &2 \cdot \binom{C}{j} \cdot 2 \cdot (j-1)! + \\
 &4|(A+E) \times C \times V| + \\
 &2\binom{C}{n} \times \binom{C}{m}
 \end{aligned}$$

where each variable is defined in Table 2. Such a formula allows us to determine the space of transformed environments that can be produced by a single transformation. As an example, we take the Data Network PDDL domain file from the International Planning Competition in 2018, and show size of novel environments when 1, 2, and 3 transformations are applied:

4. Filtering to Find *Learnable* Novelties

It is easy to construct novelties that are clearly distinct but very difficult to learn about. For example, states may be different between two environments only in some subtle way that is not visible to the agent’s sensors. Some unobservable novel event might cause an agent’s immediate destruction, allowing no response and providing no useful information the agent could use to avoid that event in the future. Novel transitions might occur only in a distant region of the state space that the

Key: Variables for PDDL+ Domain Constructs		
F is the set of functions	C is the existing set of possible conditions, where a condition is a, possibly negated, predicate or fluent statement: $ C = 2 \cdot ((3 \cdot F) + P)$. The 3 in $(3 \cdot F)$ reflects that each fluent condition may have a $<$, $>$, or $=$ comparison (we do not assume nested fluent conditions, which while possible, results in an infinite number of conditions). C^{ex} is the set of existing preconditions and effects across all existing actions and events.	
V is the set of values for a numeric range		
T is the existing set of types		
P is the existing set of predicates		
A is the existing set of actions		
E is the existing set of events		
D is the existing set of derived predicates		
U is the existing set of processes		
Key: Transformation-specific Parameters		
n : preconditions; m : effects; k : arguments for new predicate; j : upper bound on conditions		
ID	Transformation Function Signature	Number of Instances
1	changeEffectProbability(actionOrEvent, effect, value)	$ (A + E) \times C \times V $
2	changeTypeParent(typeChild, typeParent)	$ T \times T $
3	addExistingEffect(actionOrEvent, effect, probability)	$ (A + E) \times C \times V $
4	addExistingPrecondition(actionOrEvent, precondition)	$ (A + E) \times C \times V $
5	addNewAction(name, parameters, preconditions, effects)	$ \binom{C}{n} \times \binom{C}{m} $
6	addNewEvent(eventName, parameters, pre, eff, freqDist)	$ \binom{C}{n} \times \binom{C}{m} $
7	addNewPredicate(newPredicateName, arguments)	$k T $
8	addFunction(newFluentName, existingType)	$ T $
9	addNewType(newTypeName, typeParent)	$ T $
10	removePreconditionOrEffect(action/event, condition)	$ (A + E) \times C^{ex} $
11	removeAction(action)	$ A $
12	removeEvent(event)	$ E $
13	removeFluent(fluent)	$ F $
14	removeType(typeName)	$ T $
15	addDerivedPredicate(name, pred expression)	$2 \cdot \binom{C}{j} \cdot 2 \cdot (j - 1)!$
16	changeFluentEffectVal(action/event, prec/effect, val)	$ (A + E) \times C \times V $
17	changeFrequencyDistributionEvent (event, newValue)	$ E $
18	removeDerivedPredicate(derivedPredicateName)	$ D $
19	removeProcess(processName)	$ U $
20	addConstant(name, type)	$ T $
21	addProcess(processName)	1
22	changeProbability(eventName, probability)	$ E $
23	addProcessCondition(processName, condition)	$ U \times C $
24	addProcessChange(processName, change)	$ U $

Table 2. Environment Transformations.

agent never encounters in practical situations. Therefore, we consider conditions that ensure an

agent has an opportunity (and reason) to learn about an environment transformation. We say that a transformation is learnable if it changes the environment in the following ways:

Relevance: Affects an agent’s performance

Noticeability: Causes different observations

Controllability: Rewards policies differently

To formally define these qualities, we require the following definitions and assumptions.

A trajectory τ is a linked list where each element is either the empty trajectory $\emptyset$ or a tuple (a, o, τ') . Here, a is the agent’s selected action if its turn is up, or yield if it is not; o is the subsequent observation received; τ' is the remaining trajectory. We define an extension of a trajectory τ with new action a_n and observation o_n as a copy of that trajectory with a new final element $(a_n, o_n, \emptyset)$:

$$\begin{aligned} extension(\emptyset, a_n, o_n) &\equiv (a_n, o_n, \emptyset) \\ extension(\tau = (a, o, \tau'), a_n, o_n) &\equiv \\ &(a, o, extension(\tau', a_n, o_n)) \end{aligned}$$

The length of a trajectory τ is defined as 0 for $\emptyset$, and 1 + the length of the subtrajectory for all others:

$$\begin{aligned} length(\emptyset) &\equiv 0 \\ length(\tau = (a, o, \tau')) &\equiv 1 + length(\tau') \end{aligned}$$

Due to nondeterminism, an agent α with policy π and initial state s may experience any one of multiple possible histories τ of any given length. We define an agent’s decision-making policy $\pi : \tau \rightarrow A$ as a mapping from the space of trajectories to an action to take at the end of that trajectory. The probability $p(\tau|\tau^P, s, \sigma, \pi, \alpha)$ of an agent α experiencing a particular trajectory τ using policy π starting in state s with history τ^P in environment σ is given by:

$$\begin{aligned} p(\emptyset|\tau^P, s, \sigma, \pi, \alpha) &= 1 \\ p(\tau = (a, o, \tau')|\tau^P, s, \sigma = (S, A, O, Ag, Tu, \gamma, \omega), \pi, \alpha) &= \\ &\sum_{s' \in S} \gamma(s, \pi(\tau^P), s', Tu(s)) \\ &\quad * \omega^M(s', o, \alpha) \\ &\quad * p(\tau'|extension(\tau, a, o), s', \sigma, \pi, \alpha) \end{aligned}$$

We assume that an agent is motivated by some environment-independent performance measure that is known or observable to it. This function, *Performance*: $T \times S \rightarrow \mathbb{R}$ is some mapping from the space of possible trajectories T and actual states traversed to the real numbers; this function is calculated over the full sequence of states and actions in the agent’s past trajectory. The space of final states of an environment σ , $S_F(\sigma) \subset S$ is defined as the set of states which have no outgoing transitions:

$$S_F(\sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \equiv \{s \in S \mid \forall a \in A, s' \in S : \gamma(s, a, s') = 0\} \quad (2)$$

We define the **Expected Performance of a Policy (EPP)** in an environment based on a past trajectory τ^P , state history $\mathbf{s}$, and policy π as the performance of the past trajectory in any final state, or otherwise as the average expected performance of the policy over each possible next state and observation as given by recursion over the next state and updated trajectory.

$$EPP(\tau^P, \pi, \mathbf{s} = (s_0 \dots s_n \in S_F(\sigma)), \alpha, \sigma) \equiv Performance(\tau^P, s)$$

$$EPP(\tau^P, \pi, \mathbf{s} = (s_0 \dots s_n \notin S_F(\sigma)), \alpha, \sigma = (S, A, O, Ag, Tu, \gamma, \omega)) \equiv \sum_{s' \in S, o \in O} \begin{pmatrix} \gamma(s_n, \pi(\tau^P), s', \alpha) \\ * \omega(s', o, \alpha) \\ * EPP(extension(\tau^P, \pi(\tau^P), o), \pi, concat(\mathbf{s}, s'), \alpha, \sigma) \end{pmatrix} \quad (3)$$

Formally, an environment transformation $(\sigma, \sigma^M, \alpha)$ is relevant if there is some policy π that produces different expected performance starting in some state s_0 in σ and σ^M .

$$Relevant(\alpha, \sigma, \sigma^M) \equiv \exists \pi, s_0 : EPP(\emptyset, \pi, s_0, \alpha, \sigma) \neq EPP(\emptyset, \pi, s_0, \alpha, \sigma^M)$$

An environment transformation is noticeable if, from any starting state, there is some agent policy π that results in different trajectories in the modified environment σ_M than the original σ . Generally, this means that there is some set of trajectories T such that an agent will almost always experience a trajectory in T using π in σ_M , but almost never in σ . In any fully observable, deterministic environment, every environment transformation that maintains full observability and determinism is necessarily noticeable: the observation function cannot be changed without also changing the transition function, and any changed transition is immediately apparent.

$$Noticeable(\alpha, \sigma = (S, A, O, Ag, Tu, \gamma, \omega), \sigma^M) \equiv \begin{aligned} &\exists n, \pi, T : \\ &\forall \tau \in T : length(\tau) = n \\ &\wedge \sum_{s \in S^M} \sum_{\tau \in T} p(\tau | \emptyset, s, \sigma^M, \pi, \alpha) > 1 - \epsilon \\ &\wedge \sum_{s \in S} \sum_{\tau \in T} p(\tau | \emptyset, s, \sigma, \pi, \alpha) < \epsilon \end{aligned}$$

An environment transformation $(\sigma, \sigma^M, \alpha)$ is considered controllable if there is some policy π' that performs better for α in the modified environment σ^M than the optimal policy π^* for the original environment σ .

$$\begin{aligned} \text{Controllable}(\alpha, \sigma, \sigma^M) \equiv \\ \exists \pi^* \forall \pi, s \in S : \\ EPP(\emptyset, \pi, s, \alpha, \sigma) \leq EPP(\emptyset, \pi, s, \alpha, \sigma) \\ \wedge \exists \pi' \forall s \in S^M : \\ EPP(\emptyset, \pi^*, s, \alpha, \sigma') \leq EPP(\emptyset, \pi', s, \alpha, \sigma') \end{aligned}$$

We strive to approximate these conditions so as to efficiently find learnable environment transformations.

5. Related Work

Unlike work that seeks to define novelty relative to an agent’s experiences (e.g., Muhammad et al. (2021); Boulton et al. (2021); Gamage et al. (2021)), we intentionally avoid any reference to an agent’s knowledge. Such frameworks are useful because they can identify how prepared an agent is for new challenges. However, they cannot usefully describe how environment challenges differ in a way that cuts across different agents and different knowledge representations. For historical and language reasons, both currently use the term “novelty”, but in different ways.

Boulton et al. (2021) defined a theory of open world novelty where novelty occurs when an agent experiences an environment sufficiently different from its prior experiences. This is fundamentally different from our definition in several ways; first, in our framework, novelty exists independent of any given agent’s experiences or knowledge, for reasons given above. Second, our framework does not define novelty as occurring at a point in time, but rather as encompassing the time-independent difference between two environments. This means that novelty can exist with out any changes to the observation or state space. While valuable for considering what experiences could be new to a particular agent, two agents cannot truly be compared on the “same” novelty unless they have identical prior experiences; such comparisons are a key feature of our framework.

Langley (2020)’s work provides a set of broad requirements for a theory of environmental change that can help explain and measure progress in open-world learning. He suggested that such a theory would propose a formalism for environments and transformations on them. Our framework provides both, and so can be consider an example “theory” in this regard that describes environmental changes and provides specific language for characterizing how environment changes vary.

Current benchmark domains Goel et al. (2021); Xue et al. (2022); Kejriwal & Thomas (2021); Balloch et al. (2022) test an agent’s ability to respond to novelty using carefully constructed scenarios by domain experts. These testbeds provide novelties of varying difficulties and types of novelty. Almost all of these novelties are *learnable* (relevant, noticeable, and controllable) due to the human design process, yet remain limited in quantity and coverage. To our best knowledge, our testbed is the first to systematically provide infinite novelties that are not hand curated by domain experts. Instead our system randomly generates new environments and then filters them according to our learnability criteria: relevance, noticeability, and controllability.

Wiggins (2006)’s framework for creative systems in AI describes novelty as a property of a creative output which previously did not exist. While our description of environment transformation novelties makes no restriction on the generating process, it is related: part of the purpose of our framework is to describe constraints on the generation of pairs of environments pre- and post-transformation, where the later one has properties that did not previously exist in the earlier one. Thus we are broadly consistent with Wiggins’ definition.

MacGyver problems Sarathy & Scheutz (2018) are those where the goal for an agent is unreachable with the agent’s current domain model. The agent must update its model (e.g. by learning a new macro action) in order to solve the problem. When novelty occurs in the environment and an agent’s goal is no longer achievable, such a scenario can be considered a MacGyver problem. Not all novelties are MacGyver problems, since a novel environment may still permit reaching the original goal. Extending our framework to generate MacGyver problems would be possible by adding an additional filter that, given a starting state and goal, selects a new environment where that goal, from the same starting state, is no longer achievable.

6. Empirical Results

6.1 Experimental Setup

In order to test the capabilities of our novelty generator, we designed a set of experiments to determine if the generated novelties are relevant. As a test-bed, we used the data-network domain from the IPC competition. Since we were pulling from the IPC competition, we were provided with a domain file and 20 problem files. The experimental procedure is multi-stepped.

(1) First we read in the domain file for the domain, creating an objectified representation of the domain in python.

(2)Next, we look for all the possible 1-step alterations we can make on the domain that affect the transition model. This provides a total of 136 novelties for the data-network domain. There is some initial filtering during this generation stage that begins to guide the generation towards relevant novelties. The filtering at this stage looks only for new effects such that the arguments to the predicate or fluent being added can be matched to existing parameters in the action or event to which the effect is being added.

(3)After the novelties are generated, we create a set of 10 problems in the domain. These problems are created from the p01.pddl file of the IPC domain. We add in an additional set of inits to the problem such that the novel action or event is executable from the initial state. This is done to make sure that the novel action has every chance to be leveraged so we have the best chance of witnessing if the novelty is relevant.

(4) With these generated problems in hand, we then use our planning and simulation agent to test the novelty. The simulation always runs in the modified domain, however we generate 2 separate plans: one using the modified domain and one using the unmodified domain. We run both plans through the simulator, and collect the metric data. This metric value is either the defined metric for the domain, or if no metric is defined we use the number of actions executed. If the metric cost between the plan using the modified domain and the one using the unmodified domain differ for any of the 10 created problems, we say the novelty is relevant.

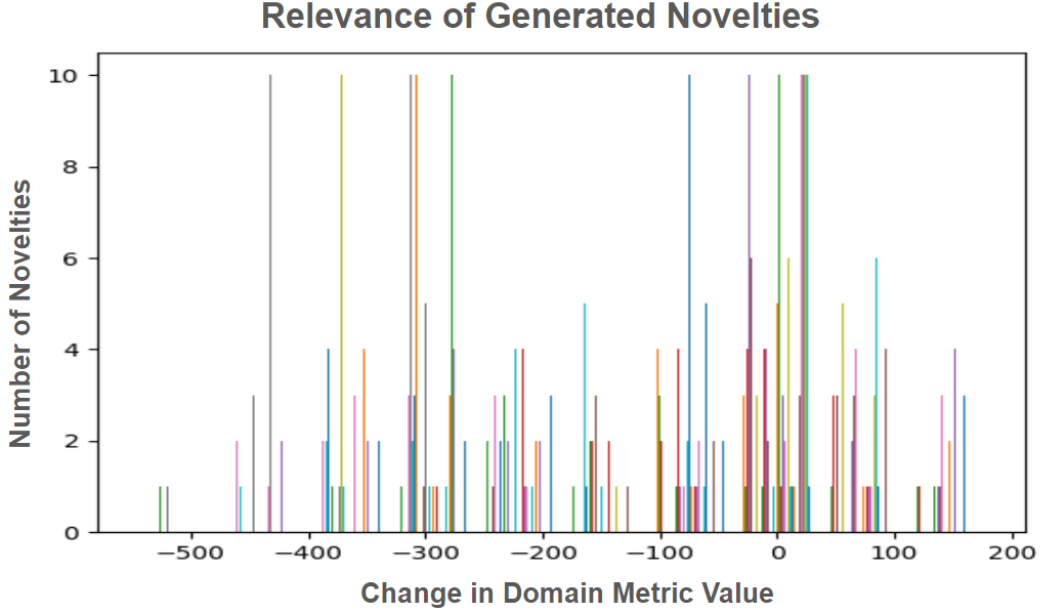


Figure 1. Showcasing the impact of novelties on the Data-Network domain from the IPC competition. The impact is measured the the change in metric value between the modified and unmodified domain

(5) After determining relevance, we perform a verification test as well as data collection on the relevant novelties. Since the problems used during the determination process were biased, we now use the provided sample problems to verify the relevance on the novelty and acquire meaningful data. Consequently, we perform step 4 again, using the given problems instead of the generated problems, saving the difference in metric score for the plans made using the modified and unmodified domain. Thus, we represent the relevance of the novelty by the average difference in metric cost seen across the given problems. These scores can be seen in Figure 1.

6.2 Discussion on Results

Analyzing Figure 1 gives us a few insights into the performance of our novelty generator. Firstly, we can see there is a wide variety of impact caused by the generated novelties. For context, the average metric cost of the unmodified domain was 647. There were a a few novelties that made the the given problems trivial to solve, a good number that made the problems easier, a lot that had a small impact, and a few that made the problems slightly harder. While this distribution is specific to this domain, it still showcases the variety of impacts provided by our generator.

Secondly, while its hard to see in this graph, none of the novelties has 0 impact on the verification problems. This means that the novelties outputted by our generator all had at least some impact on an agent acting in the domain. It should be noted however, that there are likely many relevant novelties that our generator is currently not finding.

Overall, what this means is that our novelty generator was capable of making a range of novelties that impacted agent performance in varied ways.

7. Conclusions

The work presented in this paper aimed to showcase the capability of our novelty generator to find meaningful changes to an environment description.

7.1 Future Work

While we believe important steps have been taken with generating relevant novelties, there are paths forward still to explore. While testing relevance, our current methods focused on novelties that affected the transition model. We would like to expand our filtering methods to other novelty types. Besides relevance, we would also like to collect data on the controllability of the generated novelties. Controllability in short is the ability of the agent to make use of the novelty introduced in the domain. We would also like to expand our testing to more domains, which requires a larger representation language. Currently, the language used by the novelty generator can not represent all PDDL domains, and that is a future goal.

References

- Balloch, J., Lin, Z., Hussain, M., Srinivas, A., Wright, R., Peng, X., Kim, J., & Riedl, M. (2022). Novgrid: A flexible grid world for evaluating agent response to novelty. *arXiv preprint arXiv:2203.12117*.
- Boult, T., et al. (2021). Towards a unifying framework for formal theories of novelty. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 15047–15052).
- Gamage, C., Pinto, V., Xue, C., Stephenson, M., Zhang, P., & Renz, J. (2021). Novelty generation framework for ai agents in angry birds style physics games. *2021 IEEE Conference on Games (CoG)* (pp. 1–8). IEEE.
- Goel, S., Tatiya, G., Scheutz, M., & Sinapov, J. (2021). Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds. *International Foundation for Autonomous Agents and Multiagent Systems, AAMAS*.
- Kejriwal, M., & Thomas, S. (2021). A multi-agent simulator for generating novelty in monopoly. *Simulation Modelling Practice and Theory*, 112, 102364.
- Langley, P. (2020). Open-world learning for radically autonomous agents. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 13539–13543).
- Muhammad, F., Sarathy, V., Tatiya, G., Goel, S., Gyawali, S., Guaman, M., Sinapov, J., & Scheutz, M. (2021). A novelty-centric agent architecture for changing worlds. *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 925–933).
- Sarathy, V., & Scheutz, M. (2018). Macgyver problems: Ai challenges for testing resourcefulness and creativity. *Advances in Cognitive Systems*, 6, 31–44.

FORMATTING INSTRUCTIONS

- Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19, 449–458.
- Xue, C., Pinto, V., Zhang, P., Gamage, C., Nikonova, E., & Renz, J. (2022). Science birds novelty: an open-world learning test-bed for physics domains.

A Framework for Characterizing Novel Environment Transformations in General Environments



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

A Framework for Characterizing Novel Environment Transformations in General Environments

Matthew Molineaux¹, Dustin Dannenhauer¹, Eric Kildebeck²

¹Parallax Advanced Research, Beavercreek, OH 45431
firstname.lastname@parallaxresearch.org

²University of Texas at Dallas, Richardson, TX 75080
eric.kildebeck@utdallas.edu

Abstract

To be robust to surprising developments, an intelligent agent must be able to respond to many different types of unexpected change in the world. To date, there are no general frameworks for defining and characterizing the types of environment changes that are possible. We introduce a formal and theoretical framework for defining and categorizing environment transformations, changes to the world an agent inhabits. We introduce two types of environment transformation: R-transformations which modify environment dynamics and T-transformations which modify the generation process that produces scenarios. We present a new language for describing domains, scenario generators, and transformations, called the Transformation and Simulator Abstraction Language (T-SAL), and a logical formalism that rigorously defines these concepts. Then, we offer the first formal and computational set of tests for eight categories of environment transformations. This domain-independent framework paves the way for describing unambiguous classes of novelty, constrained and domain-independent random generation of environment transformations, replication of environment transformation studies, and fair evaluation of agent robustness.

1 Introduction

A primary and unrealized goal of artificial intelligence is *robustness*: the capability to continue to perform well despite novel changes in environments an agent interacts with. Robustness is pivotal to continued autonomy in open worlds – those that are dynamic and unstable like the physical world. To date, the science of machine learning has been assumed to prepare agents to become more robust without rigorously defining what the space of such novel changes are, nor how they can be systematically varied to test the robustness of an agent. Descriptions of concept learning or classification learning problems in machine learning (Mitchell, 1997) traditionally give a set of examples from a fixed distribution, with no described temporal relationships. Reinforcement learning problems (Sutton and Barto, 2018) describe an environment that an agent receives observations from, but generally assume the environment is unchanging. Recent work on open-world novelty (Boult et al, 2021; Muhammad et al, 2021; Gamage et al, 2021), characterizes changes to an environment from the point of view of the agent, preventing direct comparisons across agents with different world views. Langley (2020) describes environmental change, but without a rigorous framework.

To better describe the challenges that a changing environment poses to an agent, we define a general class of environments and *environment transformations* (sometimes referred to as *novelties*). These definitions are useful for 1) describing the extent of environment transformations, 2) categorizing and grouping transformations with similar characteristics, 3) communicating and replicating transformations, and 4) randomly generating novelties that test the robustness of an agent to novel changes in its environment. To our knowledge, this is the first framework of definitions that describes formally and completely a space of environment transformations over a general class of environments.

Transformations to environments come in two general categories: *R-Transformations* to the dynamics of the environment, and *T-Transformations* to scenario distributions ; the latter category includes distributions over possible initial states and performance criteria. To rigorously define a general class of environments and novelties, we start with a well-known and tested formalism for representing these environments, the planning domain definition language (PDDL) family of languages. PDDL provides a general mechanism for describing individual scenarios referred to as “problems”, and a separate mechanism for describing environment dynamics referred to as “domains”. In this work, we formally describe the general notion of a “scenario generator”, which describes a distribution of scenarios rather than a single scenario.

T-SAL R-Transformations describe changes to domains as used in PDDL. PDDL domains use a subset of first-order logic to describe environment dynamics: changes that may and must happen as a result of agent actions and the natural interactions of objects without agency. Using established PDDL formalisms, we build a new language, the T-SAL (Transformation and Simulator Abstraction Language) Domain language, that covers the broadest class of environments possible. Unlike prior work on PDDL languages, we do not constrain the representation of T-SAL based on backwards compatibility or to match well the capabilities of existing automated planners; therefore, we incorporate events and processes from PDDL+ that describe continuous change, non-deterministic effects and events that describe change probabilistically, mathematical operators that describe proportional change, and an object introduction effect that describes open-world change.

T-SAL T-Transformations describe changes to scenario generators. In addition to the initial state classically described within a PDDL problem, these scenario generators describe how to draw initial state objects, and fluents from statistical distributions. Instead of a goal, scenario generators include a performance function. This is similar to existing maximization goals, but defined with respect to agents as well as the state, permitting multiple agents’ performance to be described.

T-Transformations and R-Transformations are useful both to generate and to formally describe novelty. This work contributes to the literature on open-world novelty by formally characterizing a group of novelty concepts previously described only informally. Scientists working together in the DARPA SAIL-ON program to organize and schedule program advancement developed a group of characterizations described as a “novelty hierarchy”. This novelty grouping was previously defined only informally, so some practitioners may reasonably disagree with our definitions; however, we believe they capture the essence of the designers’ intent. By providing this formal definition of the novelty hierarchy used in early novelty investigations, we hope to show that T-SAL transformations are a viable means of rigorously categorizing novelty in the future. We believe this forms a suitable basis for future investigations of the impact of novelty categories on agents.

In the following sections, we (1) formally define T-SAL domains, (2) give formulas for assessing T-SAL domain legality, (3) formally describe scenario generators, (4) formally describe transformations over T-SAL domains, (5), formally describe transformations over scenario generators, and (6) formally describe the SAIL-ON program novelty levels. A formalization for the state transition function of T-SAL is not given, but is based on PDDL+ (Fox and Long 2006) with only small modifications necessary. See Appendix C for a list of the differences between T-SAL and PDDL+.

2 Related Work

In this section, we review prior work that informed the development of our framework. In the literature on creativity, novelty is a well-defined concept. Wiggins’ (2006) Creative Systems Framework (CSF), which formally defined and extended the creativity system described by Boden (1990) in her book *The Creative Mind*, defines novelty as a property of a creative output which previously did not exist. CSF was the first formal logic-based creativity framework, describing, at the highest level, a conceptual space of universes and artifacts that naturalistic, creative agents create in a societal context. Our framework is both narrower in scope and provides greater detail, ranging over transformations to environments that consist of agents, actions, events, processes, and tasks, commonly found in AI literature. In Wiggins’ CSF, $\mathcal{R}$ denotes a set of universe constraining rules, and $\mathcal{T}$ denotes rules for traversing a space. We maintain consistency with Wiggins’ prior definitions of $\mathcal{R}$ and $\mathcal{T}$, dividing our transformations into R-Transformations and T-Transformations respectively.

The term *novelty* in AI literature has taken on at least two perspectives: novelty from the lens of an individual agent and novelty that exists from the lens of an environments’ history. Research of the former defines novelty relative to an agent’s experiences (e.g., Muhammad et al. (2021), Boulton et al. (2021), Gamage et al., (2021)), while our framework defines novelty as the latter, consistent with work on creativity (Wiggins’, 2006). Frameworks with an agent’s perspective of novelty are useful because they can identify how prepared an agent is for new challenges. However, they cannot usefully describe how environment challenges differ in a way that cuts across different agents and different knowledge representations. This limits the ability for agent-based notions of novelty to support evaluations on a variety of different AI approaches. For historical and language reasons, both areas of research currently use the term “novelty”, but in different ways; therefore we emphasize our novelty framework is environment-based.

Environment-based novelty is fundamentally different from agent-based novelty. In the theory of open world novelty by Boulton et al. (2021), novelty occurs when an agent experiences an environment sufficiently different from its prior experiences. A key feature being that novelty occurs at a point in time. In environment-based frameworks like that of Wiggins’ CSF (2006), novelty exists independent of any given agent’s experiences or knowledge. Additionally, our framework does not define novelty as occurring at a point in time, but rather as encompassing the time-independent difference between two environments. This means that novelty can exist without any changes to the observation or state space. While valuable for considering what experiences could be new to a particular agent, two agents cannot truly be compared on the “same” novelty unless they have identical prior experiences; such comparisons are a key feature of our framework.

A need for a theory of environmental change has been proposed to help explain and measure progress in open-world learning. Langley’s (2020) work provides a set of broad requirements for such a theory,

motivating a theoretical formalism for environments and transformations on them. Our framework provides both, and so can be considered an example “theory” in this regard that describes environmental changes and provides specific language for characterizing how environment changes vary.

Existing novelty generation capabilities for evaluating AI systems are designed in close alignment with simulated environments. Current benchmark domains (Goel et al. (2021), Xue et al., (2022), Kejriwal & Thomas (2021), Balloch et al., (2022)) test an agent’s ability to respond to novelty using carefully constructed scenarios by domain experts. This is current standard practice, and each class of transformations constructed applies only to a single environment. This ad-hoc construction does not permit generalization over domains or formal statements about what characteristics apply to the novel scenarios generated. To our best knowledge, our framework is the first to systematically provide infinite novelties that are not hand curated by domain experts.

Game design research has explored problems of domain generation and notions of transformations. The Metagame system (Pell, 1992) and EGGG system (Orwant, 2000) automatically generated chess-like games using a complicated system of rules and constraints designed for chess-like games in particular. Smith and Mateas (2010) decoupled the design of a game space from the exploration of that space to find interesting games. The language used to design game spaces was large and flexible compared to the prior chess-like games, but still incorporated many assumptions, such as the existence of a rectangular play space. Our framework’s language for expressing domains, based on PDDL, can model many aspects found in the game’s representations of these works; the aim of our framework is to produce novelty first and then meet other criteria later (such as being an interesting game). Therefore we focus on a more fundamental representation without worrying about domain specific knowledge beyond representational notions of objects, actions, events, and goals. To sculpt the generative space of our framework, users would provide constraints on the choice of R and T-transformations as they see fit – and such a discussion is outside the scope of this paper.

Prior work (Molineaux & Dannenhauer, 2021) defined a set of environment transformation characteristics, or “dimensions”, useful for classifying the difference between pairs of original and modified environments. These definitions are based on describing environments using a refinement of the Partially Observable Markov Decision Process formalism, rather than the relational manipulations described here. The dimension-based formalism is concerned primarily with broad characterizations of the start and endpoints rather than describing a specific pathway between them. As such, it is more difficult to verify, and does not account for human-intuitive descriptions of novelty categories such as those found in the novelty hierarchy. However, the upside there is much less flexibility in the dimension-based formalism to describe the same environments and transformations in different ways; as such, dimension values have much less “wobble room” than an arbitrary T-SAL categorization.

3 Formalism

T-SAL is a general term that refers to two languages for describing environments: the computational representation (T-SAL-CR), and the logical language that describes it, L(T-SAL), *read as ‘logic of T-SAL’*. L(T-SAL) is a first-order logic that describes domains, scenario generators, and environment transformations at an abstract level. It provides common terminology and is used for rigorous definitions. T-SAL-CR describes individual domains, using domain-specific fluents and types to describe the dynamics of domains. We build up both in this formalism; L(T-SAL) is used to define and make

general statements about environments and environment transformations. The T-SAL-CR is provided to communicate examples. An exact semantics for the T-SAL-CR is not provided here; however, it's similar to existing computational representations in the PDDL family (see Appendix C for a detailed comparison). All examples of T-SAL concepts use the well-understood Cart-Pole domain, supplemented with moving blocks and multiple carts.

3.1 T-SAL Domains

T-SAL domains describe properties and dynamics of environments. This section builds up the concept of a domain by describing simpler concepts used in the description of a domain in a bottom-up fashion. A BNF for the T-SAL domain language is given in Appendix D. All variables introduced throughout the rest of this section are shown in Table 1. Short definitions and examples of major T-SAL domain concepts are given in Table 2.

S : the set of symbols	Alt : the set of effect modifications {SET, INCREASE, DECREASE, CREATE}
V : variables in a domain description	
V : the space of variable sets	Dir : the set of change directions {INCREASE, DECREASE}
Te : terms which may be of the form of symbols, variables, integers, or reals,	Ev : the space of events
ty : a type in the space of symbols S	Ev : the space of sets of events
	ev : an event in the space of events Ev
F : the space of functions	P : the space of process models
f : a function in the space of functions F	P : the space of sets of process models
Fn : the set of function names	p : a process model in the space of process models
fn : a function name	D : the space of T-SAL domains
Cn : the space of conditions	d : a domain in the space of T-SAL domains D
Cn : the space of sequences of conditions	st : a T-SAL state that is true in the environment at a particular moment in time
cnd : a condition in the space of conditions Cn	og : an object generator
Ca : the space of calculations	OG : the space of object generators
Ca : the space of sets of calculations	vg : a value generator
calc : a calculation in the space of calculations Ca	VG : the space of value generators
Ax : the space of axioms	Va : the space of values
Ax : the space of sets of axioms	fg : a fluent generator
ax : an axiom in the space of axioms Ax	sg : a T-SAL scenario generator
E : the space of effects	SG : the space of scenario generators
E : the space of sets of effects	GF : the space of ground fluents
e : an effect in the space of effects E	gf : a ground fluent in the space of ground fluents
C : the space of continuous changes	AG : the space of agents
C : the space of sets of continuous changes	ag : an agent in the space of agents
c : a continuous change in the space of continuous changes C	t : an environment transformation
A : the space of action models	t : a sequence of environment transformations
A : the space of sets of actions models	T : the space of transformation sequences
a : an action model in the space of action models A	U : the universal set

Table 1 – Summary of variables used in L(T-SAL)

Both L(T-SAL) and T-SAL-CR borrow heavily from the PDDL (McDermott, et al. (1998); Fox & Long, (2003); Edelkamp & Hoffmann (2004); Gerevini & Long, (2005); Fox & Long, (2006); Helmert, (2008); PDDL 3.1, PDDL+) family of languages developed in the planning community, which in turn took inspiration from STRIPS (Fikes & Nilsson, 1971) and the situation calculus (McCarthy and Hayes 1950).

While we share the objective of representing environments simply and concretely, our primary objective is to enable a complete description of a range of transformation operators, which necessitates a different set of domain representation decisions. This results in a similar representation to PDDL languages that is not directly descended from any one such language. Ease of planning was not a primary design criterion, and we do not attempt to provide backwards compatibility with existing PDDL languages; instead, T-SAL follows the same guidelines but without historical baggage.

In describing T-SAL domains, we use symbols as identifiers; we refer to the set of symbols as S . Variables in a domain description come from the set $V \subset S$ (the space of variable sets is denoted $\mathbf{V}$). Another set of symbols identify particular objects; we say object o comes from the set $O \subset S$.

⌈ A domain type ty is in the space $Ty \subset S$. Five symbols are reserved for special basic types: REAL, INTEGER, BOOLEAN, AGENT, and OBJECT. A sixth standard type, POSITION, is also reserved, but is not a “basic” type. Nearly every domain requires objects and sometimes agents to be related spatially, and spatial information is useful to distinguish from non-spatial. Therefore, POSITION is reserved as a type to allow spatial properties to be automatically distinguished from non-spatial properties. The POSITION type must be specified to derive from REAL, INTEGER, or OBJECT, depending on the representation of a domain.

Definition 1: Domain Types ⌋

Example: Unique types in CartPole include CART, BLOCK, SPEED, and ANGLE.

⌈ A domain function uses the variable f and has space F and space of sets $\mathbf{F}$, has a name (*function-name*: $F \rightarrow Fn$), arguments (*function-arguments*: $F \rightarrow \mathbf{V}$) and types (*function-argument-type*: $F \times V \rightarrow Ty$), and a value type (*function-value-type*: $F \rightarrow Ty$). Function names fn come from the set $Fn \subset S$. In tuple form, a function is given as $\langle name, arguments, argument-types, value-type \rangle$.

Definition 2: Domain Functions and Function Names ⌋

Example: In T-SAL, the relationship between the cart and pole are described using roll, pitch, and yaw. The function f below describes the roll of a pole relative to its cart in T-SAL-CR:

$f: (\text{CART-VELOCITY } ?C - \text{CART}) - \text{SPEED}$

For this function, the following equivalences hold: *function-name*(f) = CART-VELOCITY; *function-arguments*(f) = [$?C$]; *function-argument-type*(f , $?C$) = CART; *function-value-type*(f) = SPEED.

⌈ Values include objects, numbers, and Boolean literals; the set of values is defined $Va \equiv O \cup \mathbb{I} \cup \mathbb{R} \cup \{\text{TRUE}, \text{FALSE}\}$. Terms include values, variables, and function terms. Function terms apply a function to other terms, including function terms. Formally, the set of terms is defined $Te \equiv V \cup Va \cup \langle Fn, \mathbf{Te} \rangle$. Here, $\mathbf{Te}$ is the space of sequences of terms from Te . Legal function terms have a term sequence the same length as the arity of the named function.

Definition 3: Values and Terms ⌋

Examples: In the T-SAL domain language representation of Cart-Pole, legal terms include 30, CART1, $?C$, (POLE-ANGLE $?C$), and (POLE-ANGLE (LEFTMOST-CART)).

Concept	Var, Space	L(T-SAL) Definition	T-SAL-CR Example
Symbol	S	<enumerated class>	CART
Variable	$v \in V$	$V \subset S$	?C
Term	Te	$Te \equiv V \cup Va \cup \langle Fn, Te \rangle$	?C; CART; 3; 1.5; TRUE; (CART-VELOCITY ?C)
Type	$ty \in Ty$	$Ty \subset S$	CART
Function	$f \in F$	$F \equiv S \times V \times (V \rightarrow S) \times S$	(CART-VELOCITY ?C – CART) – SPEED
Ground Fluent	$gf \in GF$	$GF \equiv Fn \times (V \rightarrow Va) \times Va$ Tuple: $\langle fn, args, value \rangle$	(= (CART-VELOCITY CART1) 0)
Calculation	$calc \in Ca$	$Ca \equiv Te \cup F \times Te \cup Op \times Ca \times Ca$ $\cup AggOp \times V \times Cn \times Ca$ $\cup IF \ Cn \times Ca \times Ca$ Tuples: $\langle fun, terms \rangle \langle op, calc1, calc2 \rangle$ $\langle SUM, v, con, calc \rangle \langle PRODUCT, v, con,$ $calc \rangle \langle IF, cnd, calc_T, calc_F \rangle$	0; (POLE-ANGLE ?C); (+ (POLE-ANGLE ?C) .01); (/ (SUM ?C ($\neq$ (CART-VELOCITY ?C) 0) (POLE-ANGLE ?C)) (SUM ?C ($\neq$ (CART-VELOCITY ?C) 0) 1))
Condition	$cnd \in Cn$	$Cn \equiv Te \cup PC \cup BOp \times Cn \times Cn$ $\cup FORALL \times V \times Cn \times Cn$ $\cup HAS-VALUE \times V \times Ca$ Tuples: $\langle AND, cnd1, cnd2 \rangle \langle OR, cnd1, cnd2 \rangle$ $\langle FORALL, v, cons, req \rangle \langle HAS-VALUE, v, calc \rangle$	TRUE; (< (POLE-ANGLE ?C) 30); (AND (< (POLE-ANGLE CART1) 30) (> (POLE- ANGLE CART1) -30)); (FORALL ?C ($\neq$ (CART-VELOCITY ?C) 0) (< (POLE-ANGLE ?C) 30)); (HAS-VALUE ?A (POLE-ANGLE ?C))
Effect	$e \in E$	$E \equiv S \times V \times Alt \times Te \times [0..1]$	(DECREASE (POLE-ANGLE ?C) 0.1) [0.5]
Continuous Change	$c \in C$	$C \equiv S \times V \times Dir \times Ca$	(INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C))
Action Model	$a \in A$	$A \equiv S \times Te \times V \times (V \rightarrow Ty) \times Cn \times E$ Tuple: $\langle name, performer, args, argTypes,$ $prec, effs \rangle$	(:ACTION PUSH :PERFORMER ?AG :PARAMETERS (?C – CART) :PRECONDITIONS ((CONTROLS ?AG ?C) (= (AGENT-FORCE) ?AG)) :EFFECTS ((INCREASE (CART-VELOCITY ?C) ?FORCE)))
Event Model	$ev \in Ev$	$Ev \equiv S \times V \times (V \rightarrow Ty) \times [0..1] \times \mathbb{R}^{\geq 0} \times Cn \times E$ Tuple: $\langle name, quals, qualTypes, prob, freq,$ $triggers, effs \rangle$	(:EVENT FINISHES :QUALITIES (?C – CART) :TRIGGERS ((UPRIGHT ?C) (< (CART-VELOCITY ?C) 0.1) (> (CART-VELOCITY ?C) -0.1) (CONTROLS ?AG ?C)) :EFFECTS (WINS ?AG))
Process Model	$p \in P$	$P \equiv S \times V \times (V \rightarrow Ty) \times Cn \times C$ Tuple: $\langle name, quals, qualTypes, conds,$ $changes \rangle$	(:PROCESS POLE-ANGLE-CHANGES :QUALITIES (?C – CART) :CONDITIONS (FORALL ?AG TRUE (NOT (WINS ?AG))) :EFFECTS (AND (INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C))))))

Table 2 – Summary of key T-SAL domain concepts with definitions and examples

Conditions (variable *cnd*, space *Cn*, space of sequences **Cn**) specify complex requirements over states using comparisons and Boolean operators. Like calculations, conditions have the form of a term, a comparison of two values, or a combined calculation in which two operands are combined by a Boolean operation: $Cn \equiv Te \cup BOp \times Cn \times Cn \cup \text{FORALL} \times V \times Cn \times Cn \cup Ineq \times Ca \times Ca \cup \text{NOT} \times Cn$. Supported Boolean operations are $BOp \equiv \{\text{AND}, \text{OR}\}$. Supported comparison operators are the inequalities $Ineq \equiv \{=, \neq, <, >, \leq, \geq\}$.

Definition 4: Conditions

Examples: In the T-SAL knowledge representation of Cart-Pole, legal conditions include:

- ($< (\text{POLE-ANGLE } ?C) 30$)
 - True iff the pole on the cart bound to ?C has roll less than 30
- ($\text{AND} (< (\text{POLE-ANGLE CART1}) 30) (> (\text{POLE-ANGLE CART1}) -30)$)
 - True iff the pole on the object CART1 has angle between -30 and 30
- ($\text{AND} (<= (\text{POLE-ANGLE CART1}) (\text{POLE-ANGLE CART2}))$)
 - True iff the pole on the object CART1 has a smaller angle than that on CART2.
- ($\text{FORALL } ?C (!= (\text{CART-VELOCITY } ?C) 0) (< (\text{POLE-ANGLE } ?C) 30)$)
 - True iff every pole with non-zero velocity also has pole angle less than 30

Calculations (variable *calc*, space *Ca*, space of sets **Ca**) are functional expressions composed of terms and operators. Calculation operators include basic arithmetic operations, distribution draws, aggregations over sets, and the trinary if operator. Where terms have the semantics of a lookup in a database incorporating state information, calculations describe executed procedures that return values. A calculation must be a term, a combined calculation in which two operands are combined by an operation, an aggregation over a set of variables, or a trinary if that returns the result of one of two sub-calculations: $Ca \equiv Te \cup Op \times Ca \times Ca \cup AggOp \times V \times Cn \times Ca \cup \text{IF } Cn \times Ca \times Ca$. Supported operations are $Op \in \{+, -, *, /, :UNIFORM, :GAUSSIAN\}$. These operations have the usual definitions, and all require that their operands are numeric (i.e., derived from the type REAL or INTEGER). The +, -, *, and / operations yield a value of type REAL if either operand is REAL, or INTEGER otherwise. The :UNIFORM operation draws a value from the uniform distribution, with a resulting value in the range defined by its operands (the first operand providing the lower bound, and the second the upper bound, inclusive). The operands and the resulting value must have the INTEGER type. The :GAUSSIAN operation draws from the gaussian distribution; its operands are interpreted as a mean and standard distribution, and the type of the calculation will be REAL. Aggregation operations are $AggOp \in \{\text{SUM}, \text{PRODUCT}\}$. Aggregations have a single variable, constrained to meet a condition; the sub-calculation is calculated for every permissible value of the variable and results are aggregated. Conditional calculations give a condition that determines which one of a pair of calculations to evaluate; if the condition evaluates to TRUE, the first calculation is evaluated, and if FALSE the second.

Definition 5: Calculations

Example: In the T-SAL knowledge representation of Cart-Pole, legal calculations include:

- ($\text{POLE-ANGLE } ?C$)
- ($+ (\text{POLE-ANGLE } ?C) .01$)
- ($/ (\text{SUM } ?C (\neq (\text{CART-VELOCITY } ?C) 0) (\text{POLE-ANGLE } ?C)) (\text{SUM } ?C (\neq (\text{CART-VELOCITY } ?C) 0) 1)$)
 - Average angle of poles with nonzero velocity

⌈ Axioms define the truth of fluents in a state in terms of a condition. A fluent defined by axioms must have value type **BOOLEAN** and takes on the value **TRUE** when the associated condition is satisfied, or **FALSE** otherwise. No function may therefore both have an axiom definition and appear in action or event effects. Axioms use the variable ax (Ax for the space, **Ax** for the space of sets), and describes a function name (*axiom-name*: $Ax \rightarrow S$), argument types (*axiom-argument*: $Ax \times V \rightarrow S$), and an antecedent condition (*axiom-antecedent*: $Ax \rightarrow Cn$).

Definition 6: Axioms ┘

Example: In Cart-Pole, an axiom ax could say that a “live” cart is one whose pole angles are less than 30:

```
(:- (UPRIGHT ?C - CART)
    (AND (< (POLE-ROLL ?C) 0.1) (> (POLE-ANGLE ?C) -0.1)))
```

Here, *axiom-name*(ax) = UPRIGHT; *axiom-argument*(ax , ?C) = CART; and *axiom-antecedent*(ax) = $\langle \text{AND}, \langle \text{<}, \text{POLE-ANGLE}, [\text{?C}], 0.1 \rangle \langle \text{>}, \text{POLE-ANGLE}, [\text{?C}], -0.1 \rangle \rangle$

⌈ An effect, which uses the variable e and has space E (**E** for the space of sets), describes an instantaneous alteration to the world caused by an action or effect transition. Mostly, these change the values of fluents. However, a second type of effect creates a new object; this is necessary in general to allow us to describe open worlds where objects can proliferate without a bound. An effect has a symbol that names the function it affects (*effect-name*: $E \rightarrow Fn$), assignments to that function’s arguments (*effect-argument*: $E \times V \rightarrow Te$), a modification type (*effect-modification*: $E \rightarrow Alt$), post-effect value (*effect-value*: $E \rightarrow Te$), and probability (*effect-probability*: $E \rightarrow [0..1]$). The set of possible alterations is given by: $Alt \equiv \{\text{SET}, \text{INCREASE}, \text{DECREASE}, \text{CREATE}\}$. In the special case of a **CREATE** alteration, the effect’s name is the type symbol of the object created rather than that of a function, and the effect’s value describes a name prefix to use in referring to the new object.

Definition 7: Effects ┘

Example: In Cart-Pole, an example legal effect e is (DECREASE (POLE-ANGLE ?C) 0.1) [0.5], which states that the value of the fluent POLE-ANGLE for the object bound to ?C decreases by 0.1 during a transition. Here, *effect-name*(e) = POLE-ANGLE; *effect-argument*(e , ?C) = ?C; *effect-modification*(e) = DECREASE; *effect-value*(e) = 0.1; and *effect-probability*(e) = 0.5.

A **CREATE** effect of the form (CREATE BLOCK ?B “new-block”) causes a new block to exist in the Cart-Pole environment. Here, *effect-name*(e) = BLOCK; *effect-argument*(e , BLOCK) = ?B; *effect-modification*(e) = CREATE; and *effect-value*(e) = “new-block”. With no probability explicitly described, we give *effect-probability*(e) = 1.

⌈ A continuous change, which uses the variable c and has space C (**C** for the space of sets), describes continuous time changes to fluents. A change describes the fluent it changes with a name symbol (*change-name*: $C \rightarrow S$), argument assignments to that fluent (*change-argument*: $C \times V \rightarrow Te$), direction of change (*change-direction*: $C \rightarrow Dir$) and a derivative calculation (*change-derivative*: $C \rightarrow Ca$). Here, the set of possible directions is given by: $Dir \equiv (\text{increase}, \text{decrease})$. A special DT symbol is reserved to represent the time differential in continuous change derivatives only. Each continuous change must use this symbol once.

Definition 8: Continuous Changes ┘

Example: In Cart-Pole, an example legal change c is $(\text{INCREASE}(\text{POLE-ANGLE } ?C) (* \text{DT}(\text{ANGULAR-MOTION } ?C)))$, which states that the value of the fluent POLE-ANGLE for the cart bound to $?C$ increases linearly with time according to the angular motion function (not described further here). Here, $\text{change-name}(c) = \text{POLE-ANGLE}$; $\text{change-argument}(c, ?C) = ?C$; $\text{change-direction}(c) = \text{INCREASE}$; and $\text{change-derivative}(c) = (* \text{DT}(\text{ANGULAR-MOTION } ?C))$.

▮ Action models, event models, and process models describe when and how the world changes. Actions represent change caused by a performer; a performer can choose whether to take an action and with what parameters. Some actions may act differently for different performers or not be available to certain performers, so a performer variable gives a reference to the performer for preconditions and effects. Formally, an action model $a \in A$ (space of sets **A**) has a name ($\text{action-name}: A \rightarrow S$), performer ($\text{action-performer}: A \rightarrow Te$), parameters ($\text{action-parameters}: A \rightarrow V$), parameter types ($\text{action-parameter-type}: A \times V \rightarrow Ty$), preconditions ($\text{action-preconditions}: A \rightarrow \mathbf{Cn}$), and effects ($\text{action-effects}: A \rightarrow E$). Tuple notation for actions is $\langle \text{name}, \text{performer}, \text{args}, \text{argTypes}, \text{precs}, \text{effs} \rangle$; a tuple in this form can be interpreted as an action that can respond to each above function.

Definition 9: Action Models ▮

Example: An action model a in cart pole pushes the cart forward, instantaneously increasing its velocity in the x direction:

```
(:ACTION PUSH
:PERFORMER ?AG
:PARAMETERS (?C – CART)
:PRECONDITIONS ((CONTROLS ?AG ?C) (= (AGENT-FORCE ?AG) ?FORCE)
:EFFECTS ((INCREASE (CART-VELOCITY ?C) ?FORCE))
```

Here, $\text{action-name}(a) = \text{PUSH}$; $\text{action-performer}(a) = ?AG$; $\text{action-parameters}(a) = [?C]$; $\text{action-parameter-type}(a, ?C) = \text{CART}$; $\text{action-preconditions}(a) = [(= (\text{AGENT-FORCE } ?AG) ?FORCE) (\text{CONTROLS } ?AG ?C)]$; and $\text{action-effects}(a) = [(\text{INCREASE } (\text{CART-VELOCITY } ?C) 1)]$.

▮ Event models represent instantaneous change not caused by a modelled agent; they have no parameters, because no entity (performer or otherwise) chooses an event to happen. However, qualities refer to a set of variables that may be instantiated within the triggers of the event and aid in understanding. All events are triggered in one of three ways: occur 1) by default, they must occur immediately when their triggers are met; 2) if an event model has a probability between 0 and 1, modelled events occur probabilistically when their triggers are met; 3) if an event model has a non-zero frequency, modelled events occur at random times, with interarrival times described by a poisson distribution with the modelled frequency. No event model can have both a probability less than 1 and a non-zero frequency. An event $ev \in Ev$ (space of sets **Ev**) has a name ($\text{event-name}: Ev \rightarrow S$), qualities ($\text{event-qualities}: Ev \rightarrow V$), quality types ($\text{event-quality-type}: Ev \times V \rightarrow Ty$), probability ($\text{event-probability}: Ev \rightarrow [0..1]$), frequency ($\text{event-frequency}: Ev \rightarrow \mathbb{R}^{\geq 0}$), triggers ($\text{event-triggers}: Ev \rightarrow \mathbf{Cn}$), and effects ($\text{event-effects}: Ev \rightarrow E$). Tuple notation for events is $\langle \text{name}, \text{quals}, \text{qualTypes}, \text{prob}, \text{freq}, \text{triggers}, \text{effs} \rangle$; a tuple in this form can be interpreted as an event that can respond to each above function.

Definition 10: Event Models ▮

Example: An event ev in cart pole causes a win for a player whose cart reaches an angle and velocity within a certain range:

```
(:EVENT FINISHES
:QUALITIES (?C – CART)
:TRIGGERS ((UPRIGHT ?C) (< (CART-VELOCITY ?C) 0.1) (> (CART-VELOCITY ?C) -0.1) (CONTROLS ?AG ?C))
:EFFECTS (WINS ?AG))
```

Here, $event-name(ev) = FINISHES$; $event-probability(ev) = 1$; $event-frequency(ev) = 0$; $event-qualities(ev) = [?C]$; $event-quality-type(ev, ?C) = CART$; $event-triggers(ev) = [(= (UPRIGHT ?C) TRUE), (< (CART-VELOCITY ?C) 0.1), (> (CART-VELOCITY ?C) -0.1), (= (CONTROLS ?AG ?C) TRUE)]$; and $event-effects(ev) = [(SET (WINS ?AG) TRUE)]$.

⌈ Process models represent continuous change over time. Like events, they have qualities and are not controlled by any agent. Unlike events (or actions), they represent change over time, and describe continuous changes that occur over the duration of the process rather than effects. A process will continue for as long as its conditions are met. A process $p \in P$ (space of sets P) has a name ($process-name: P \rightarrow S$), qualities ($process-qualities: P \rightarrow V$), quality types ($process-quality-type: P \times V \rightarrow Ty$), conditions ($process-conditions: P \rightarrow Cn$), and continuous changes ($process-changes: P \rightarrow C$). Tuple notation for processes is $\langle name, quals, qualTypes, conds, changes \rangle$; a tuple in this form can be interpreted as a process model that can respond to each above function.

Definition 11: Process Models ┘

Example: A process p in cart pole causes the pole to change position:

```
(:PROCESS POLE-ANGLE-CHANGES
:QUALITIES (?C – CART)
:CONDITIONS (FORALL ?AG TRUE (NOT (WINS ?AG)))
:EFFECTS (AND (INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C)))))
```

Here, $process-name(p) = POLE-ANGLE-CHANGES$; $process-qualities(p) = [?C]$; $process-quality-type(p, ?C) = CART$; $process-conditions(p) = [(FORALL ?AG TRUE (= (WINS ?AG) FALSE))]$; and $process-changes(p) = [(INCREASE (POLE-ANGLE ?C) (* DT (ANGULAR-MOTION ?C)))]$.

⌈ A T-SAL domain, which we use the variable d for (D is the space of possible domains), includes types ($domain-types: D \rightarrow S$), supertypes ($domain-supertypes: D \rightarrow S \times S$), named constants with given types ($domain-constants: D \rightarrow S \times S$), functions ($domain-functions: D \rightarrow F$), axioms ($domain-axioms: D \rightarrow Ax$), actions ($domain-actions: D \rightarrow A$), events ($domain-events: D \rightarrow E$), and processes ($domain-processes: D \rightarrow P$). Tuple notation for domains is $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle$.

Definition 12: T-SAL Domains ┘

Legality of a T-SAL domain, based mainly on type agreement, is described in Appendix A.

3.2 T-SAL States, Goals, and Scenario Generators

In addition to dynamics, T-SAL describes environments in terms of distributions of starting states and tasks. In this section, we define T-SAL scenario generators, which define both of these. To describe a

scenario generator, we need the concepts of state, and various simpler generator functions: object generators, value generators, and fluent generators.

⌈ A T-SAL state st describes what is true in the environment at a particular instant in time. We describe a state with a tuple $\langle objects, defaults, assignments \rangle$. Each T-SAL state is associated with a T-SAL domain $d = \langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle$, and defines the current set of objects belonging to every type in $types$ and a value for every possible fluent grounding using $functions$ with those objects. State objects ($state-objects: St \times S \rightarrow S$) includes all objects of every type in the domain. The defaults ($state-defaults: St \times F \rightarrow Te$) give a default value for every function, which is assigned by this state to any fluent grounding that is not given a specific value in $assignments$. The state's assignments ($state-assignments: St \rightarrow GF$) give specific values for a list of ground fluents. Ground fluents use the variable gf , space GF and space of sequences GF . Each ground fluent names the function the fluent is from ($ground-fluent-function-name: GF \rightarrow Fn$), argument values ($ground-fluent-argument: GF \times V \rightarrow Va$) and a fluent value ($ground-fluent-value: GF \rightarrow Va$). Tuple form for ground fluents is $\langle function-name, argument-values, fluent-value \rangle$.

Definition 13: T-SAL State ┘

Example: In a simple Cart-Pole state st there is one cart, CART1, and three blocks, BLOCK1, BLOCK2, and BLOCK3. A single agent, AGENT1 exists. Other types (speed, position, angle) are derived from REAL and are non-enumerated. Fluents in the domain include $f1 = (CART-VELOCITY ?C - CART) - SPEED$, $f2 = (CART-POSITION ?C - CART) - POSITION$, $f3 = (POLE-ANGLE ?C - CART) - ANGLE$, $f4 = (BLOCK-POSITION ?B - BLOCK) - POSITION$, $f5 = (BLOCK-VELOCITY ?B - BLOCK) - SPEED$, $f6 = (AGENT-FORCE ?AG - AGENT) - REAL$, $f7 = (CONTROLS ?AG - AGENT ?C - CART) - BOOLEAN$, and $f8 = (WINS ?AG - AGENT) - BOOLEAN$. Relevant values are: $state-objects(st, CART) = [CART1]$; $state-objects(st, BLOCK) = [BLOCK1, BLOCK2, BLOCK3]$; $state-defaults(st, f1) = 0$; $state-defaults(st, f3) = 0.1$; $state-defaults(st, f5) = 0$; $state-defaults(st, f6) = 1$; $state-defaults(st, f7) = FALSE$; $state-defaults(st, f8) = FALSE$; $state-assignments(st) = [CART-POSITION(CART1) = 0, BLOCK-POSITION(BLOCK1) = 5, BLOCK-POSITION(BLOCK2) = 5, BLOCK-POSITION(BLOCK3) = 10, CONTROLS(AGENT1, CART1) = TRUE]$.

Note that every legal assignment to every fluent can be valued using either a default or an explicit assignment in st . In this case, functions $f2$ and $f4$, which have no defaults, must be fully enumerated, as shown.

⌈ An object generator describes a distribution of objects in the environment. Whereas a T-SAL domain refers to certain “constant” objects present in every scenario, object generators describe the objects that are present in an environment as part of a particular scenario. An object generator og has a name ($object-generator-name: OG \rightarrow S$), type ($object-generator-type: OG \rightarrow S$), and a draw function that yields a set of distinct objects when called ($object-generator-draw: OG \rightarrow (\rightarrow S)$). As randomness is frequently a desirable property of such generators, the draw function may return sets with different contents and sizes on subsequent invocations. A set of basic draw functions is described in Section 3.4.3.

Definition 14: Object Generator ┘

Example: A simple object generator *og* makes 5 block objects. We write this like so:

```
OBJECTGENERATOR(blockGroup, BLOCK, OBJECTLIST(5, "block"))
```

The standard draw function `OBJECTLIST` is defined to return N (the first argument) object names, each starting with the prefix given by the second argument. The object generator *og* has the following characteristics: $object-generator-name(og) = cartGroup$; $object-generator-type(og) = CART$; $object-generator-draw(og) = OBJECTLIST(5, "cart")$.

⌈ A value generator returns values from a random distribution; these values may be generated by from sets defined by object generators, domain constants, or from the reals, integers, or booleans. A value generator *vg* has a name ($value-generator-name: VG \rightarrow S$) and provides a draw function that yields a sequence of values when called ($value-generator-draw: VG \rightarrow (\rightarrow \mathbf{Va})$). The draw function may produce random results or static results; a set of basic value generator draw functions is given in Section 3.4.4.

Definition 15: Value Generator ┘

Example: A simple value generator *vg* for positions gives a single random real value between -20 and 20. We write this like so:

```
VALUEGENERATOR(RANDOMPOSITION, UNIFORMREALDISTRIBUTION(-20, 20))
```

This uses a standard draw function that pulls from a uniform distribution. The value generator *vg* has the following characteristics: $value-generator-name(vg) = RANDOMPOSITION$; $value-generator-draw(vg) = UNIFORMREALDISTRIBUTION(-20, 20)$.

⌈ A fluent generator describes a distribution of fluent values for a particular domain function. A fluent generator *fg* has a function name ($fluent-generator-function-name: FG \rightarrow Fn$) that it generates fluents for, and provides a draw function ($fluent-generator-draw: FG \rightarrow (\rightarrow \mathbf{GF})$) that yields a list of ground fluents describing individual fluent groundings; these do not necessarily map all possible assignments to the named function. Basic fluent generators are described in Section 3.4.5.

Definition 16: Fluent Generator ┘

Example: A simple fluent generator *fg* for assigning block positions gives a random real value between -20 and 20 to each block. We write this like so:

```
FLUENTGENERATOR(BLOCK-POSITION, ALLPERMUTATIONS([BLOCKGROUP], RANDOMPOSITION))
```

The standard draw function, `ALLPERMUTATIONS`, described here creates a ground fluent for every permutation of unique objects generated by the object generators given as its first argument, and assigns each a value generated by the value generator given as its second argument. The fluent generator has the following characteristics: $fluent-generator-function-name(fg) = BLOCK-POSITION$; $fluent-generator-draw(fg) = ALLPERMUTATIONS([BLOCKGROUP], RANDOMPOSITION)$.

⌈ A T-SAL scenario generator *sg* is a tuple $\langle fluents, defaults, object-generators, value-generators, fluent-generators, performance \rangle$, that describes a distribution over environment starting states, as well as a performance calculation that describes the success of each agent. The first item, *fluents*, (*scenario-*

generator-object-generators: $SG \rightarrow \mathbf{GF}$), gives a set of ground fluents present in every scenario generated; these are not randomized. The second item, *defaults*, (*scenario-generator-defaults*: $SG \times Fn \rightarrow Va$) maps each function name to a default value that can be assumed in the absence of contradictory assignments (from the fluents list and fluent generators). Each fluent in the corresponding domain should be covered by this mapping. The third item, *object-generators*, (*scenario-generator-object-generators*: $SG \rightarrow \mathbf{OG}$) sets up objects present in a starting state. The fourth item, *value-generators*, (*scenario-generator-value-generators*: $SG \rightarrow \mathbf{VG}$) gives a set of value generators defined for use by the fluent generators. The fifth scenario generator item, *fluent-generators*, (*scenario-generator-fluent-generators*: $SG \rightarrow \mathbf{FG}$) gives a set of fluent generators that describe distributions over fluents in the starting state. Finally, the last item, *performance*, gives a performance calculation as a calculation (*scenario-generator-performance-calculation*: $SG \rightarrow Ca$). This calculation must have a single free variable, ?AG, which identifies the agent for which performance is to be computed. For simplicity, we assume that this calculation is always intended to be maximized and can be evaluated in any state.

Definition 17: T-SAL Scenario Generator

Example: A scenario generator *sg* defines cart-pole performance as the negative sum of the squared angles of all poles controlled by that agent.

$$\text{scenario-generator-performance-calculation}(sg) = \\ (\text{SUM } ?C (\text{CONTROLS } ?AG ?C) (- (* (\text{POLE-ANGLE } ?C) (\text{POLE-ANGLE } ?C))))$$

To generate a scenario, a new state object is created with defaults from the state generator, objects created by drawing from each object-generator, and assignments created by drawing from each fluent generator. Value generators are used indirectly during fluent generators draws. Finally, fluents in the scenario generator fluents list are added to the state, overwriting any generated and/or default fluents with the same fluent name and arguments.

An environment consists of a domain d and a scenario generator *sg*. To be legal, *sg* must have a close relationship to the domain d :

- *fluents* must have only ground fluents with appropriate arity and values of appropriate types
- *defaults* must give an appropriately typed value for every function in d
- *object-generators* must use object types in d
- *fluent-generators* must always generate legal ground fluents for the described function, which must also be in domain d
- *performance* must be a real-valued calculation with a single free variable, ?AG, referring to the target agent, it must only reference functions in the domain d , and all symbols must reference constants from the domain d .

Since the returns from generator draw functions are defined to vary over time, constraints on the draw function outputs are expressed below using the “G” temporal operator, which means “always”.

3.3 T-SAL Transformations and Transformation Sequences

⌈ We use the variable t to describe an environment transformation, given as a tuple with a head and named arguments. The space of possible environment transformations is denoted T . We will also refer to a transformation sequence $\mathbf{t} = [t_0, \dots, t_n]$, with the transformation sequence space denoted as $\mathbf{T}$. A transformation has a type (*transformation-type*: $T \rightarrow S$) and arguments (*transformation-argument*: $T \times S \rightarrow U$). Here, U denotes the universal set; the meaning of particular transformation arguments is dependent on the type. Individual transformations are defined in Section 3.5.2 (T-Transformations) and Appendix B (R-Transformations).

Definition 18: Transformations and Transformation Sequences ⌋

Example: A transformation t to a cart-pole environment that subtracts push force from a budget is given as:

ADDACTIONEFFECT(ACTIONNAME: PUSH, EFFECT: (DECREASE (PUSH-BUDGET ?AG) ?FORCE))

The transformation t has *transformation-type*(t) = ADDACTIONEFFECT. The first argument is referred to by *transformation-argument*(t , ACTIONNAME) = PUSH.

⌈ Applying a transformation to an environment yields a new environment: *apply*: $T \times D \times SG \rightarrow D \times SG$. We also define a function that applies a sequence of transformations (*applySequence*: $\mathbf{T} \times D \times SG \rightarrow D \times SG$) as follows.

$applySequence(\emptyset, d, sg) \equiv \langle d, sg \rangle$

$applySequence(\mathbf{t} = [t_0, \dots, t_n], d, sg) \equiv applySequence([t_1, \dots, t_n], apply(t_0, d, sg))$.

Definition 19: Transformation and Transformation Sequence Application ⌋

3.4 T-SAL R-Transformations

R-transformations are modifications to the environment dynamics. We formally define 30 transformations on T-SAL environments in Appendix A with a summary of these given here in Table 3. State space transformations are those that change the set of states the environment may contain. Performer control transformations change a performer's ability to interact with the environment. Instantaneous environment change transformations change all non-voluntary instantaneous transitions. Durative natural change transformations change processes which operate over continuously over time.

Summary of all R-transformations			
State Space Transformations	Performer Control Transformations	Instantaneous Environment Change Transformations	Durative Natural Change Transformations
ADDTYPE	ADDACTION	ADDEVENT	ADDPROCESS
ADDTYPEPARENT	ADDPRECONDITION	CHANGEFREQUENCY	ADDPROCESSCONDITION
ADDCONSTANT	ADDACTIONEFFECT	CHANGEPROBABILITY	ADDPROCESSCHANGE
ADDFUNCTION	REMOVEACTION	ADDTRIGGER	REMOVEPROCESS
ADDAXIOM	REMOVEPRECONDITION	ADDEVENTEFFECT	REMOVEPROCESSCONDITIONS
REMOVETYPE	REMOVEACTIONEFFECT	REMOVEEVENT	REMOVEPROCESSCHANGE
REMOVETYPEPARENT		REMOVETRIGGER	
REMOVECONSTANT		REMOVEEVENTEFFECT	
REMOVEFUNCTION			
REMOVEAXIOM			

Table 3: The *apply* function is defined with respect to all these R-Transformations within Appendix A.

3.5 T-SAL T-Transformations

T-Transformations are modifications to the scenario generator. We define a set of T-Transformations which cover the space of possible generator modifications below; first, we give examples for a paradigmatic domain, then we define the particular transformations possible, as well as the sub-generators that can be used.

To easily transform scenario generators, it will be useful to supersede existing value or object generators. Therefore, when we add a new generator, it should be assumed to replace any existing generator with the same name, and all references in other generators to the old generator become references to the new generator. As such, we need only five types of T-Transformation: `ADDFLUENTVALUE`, `ADDDFAULTVALUE`, `ADDOBJECTGENERATOR`, `ADDVALUEGENERATOR`, `ADDFLUENTGENERATOR`, and `ADDPERFORMANCECALCULATION`. As they are particularly novel, we give additional examples below of T-Transformations for value, object, and fluent generators useful in various domains; we follow this with a formal definition of these five transformations, and a description of the value, object, and fluent generators used.

3.5.1 T-Transformation examples

In Mudworld (Molineaux and Aha 2014), a set of rovers attempt to travel to a set of destinations on a 6x6 grid. Any location may be muddy, with a 30% chance. Mud slows the rover down to half speed, with some variability across scenarios. All destinations are within four cell moves. The following T-Transformations construct a scenario generator for this environment.

```

ADDVALUEGENERATOR(XLoc, UNIFORMINTEGERDISTRIBUTION(1, 6))
ADDVALUEGENERATOR(YLoc, UNIFORMINTEGERDISTRIBUTION(1, 6))
ADDVALUEGENERATOR(Xset, INTEGERSEQUENCE(1, 6))
ADDVALUEGENERATOR(Yset, INTEGERSEQUENCE(1, 6))
ADDOBJECTGENERATOR(robotGroup, Robot, OBJECTLIST(6, "rover"))
ADDVALUEGENERATOR(RobotG, DRAWFROMOBJECTSET(robotGroup))
ADDVALUEGENERATOR(RandRobot, DRAWALLFROMOBJECTSET(robotGroup))
ADDVALUEGENERATOR(TRUE, CONSTANTFUNCTION(TRUE))

```

```

ADDVALUEGENERATOR(IsMuddy, BERNOULLIDISTRIBUTION(0.3))
ADDFLUENTGENERATOR("muddy", ALLPERMUTATIONS([Xset, Yset], IsMuddy))
ADDVALUEGENERATOR(SpeedInMud, GAUSSIANDISTRIBUTION(0.5, 0.2))
ADDFLUENTGENERATOR("speed-in-mud", ALLPERMUTATIONS([], SpeedInMud))
ADDVALUEGENERATOR(ClosePos, FILTER(
    DRAWTUPLE([Xloc, Yloc, Xloc, Yloc], ["x1", "y1", "x2", "y2"])),
     $\lambda(x_1, y_1, x_2, y_2) \rightarrow \{|x_1 - x_2| + |y_1 - y_2| < 4\}$ )))
ADDVALUEGENERATOR(CloseSet, NEWSET(NDRAWS(ClosePos, 6))
ADDFLUENTGENERATOR("robot-x-loc", NFLUENTDRAWS([Robot], CloseSet.x1, 6))
ADDFLUENTGENERATOR("robot-y-loc", NFLUENTDRAWS([Robot], CloseSet.y1, 6))
ADDFLUENTGENERATOR("robot-dest", NFLUENTDRAWS([Robot, CloseSet.x2, CloseSet.y2], TRUE, 6))

```

Here, we consider a scenario generator that creates a multi-dimensional random value for a camera's starting location:

```

ADDVALUEGENERATOR(Position, NDIMENSIONALGAUSSIANDISTRIBUTION([0.5, 0.7, 0.2], [0.1, 0.2, 0.05]))
ADDFLUENTGENERATOR("camera-starting-position",
    NFLUENTDRAWS([Position.x1, Position.x2, Position.x3], True, 1))

```

3.5.2 T-Transformation Definitions

We define the apply function for five T-Transformations:

```

apply(ADDFLUENTVALUE(FUNCTIONNAME: name, FLUENTARGS: args, VALUE: value), d,
    {fluents, defaults, object-generators, value-generators, fluent-generators, performance}) ≡
    {({fname, fargs, fvalue} | {fname, fargs, fvalue} ∈ fluents ∧ (fname ≠ name ∨ fargs ≠ args)}
    ∪ {{name, args, value}},
    defaults, object-generators, value-generators, fluent-generators)

apply(ADDDEFAULTVALUE(FUNCTIONNAME: name, VALUE: value), d,
    {fluents, defaults, object-generators, value-generators, fluent-generators, performance}) ≡
    {fluents, {{fname, fvalue} | {fname, fvalue} ∈ defaults ∧ fname ≠ name} ∪ {{name, value}},
    object-generators, value-generators, fluent-generators)

apply(ADDOBJECTGENERATOR (NAME: name, TYPE: type, DRAWFUNCTION: drawFn), d,
    {fluents, defaults, object-generators, value-generators, fluent-generators, performance}) ≡
    {fluents, defaults,
    {og ∈ object-generators | object-generator-name(og) ≠ object-generator-name(newGenerator)}
    ∪ {{name, type, drawFn}},
    value-generators, fluent-generators)

apply(ADDVALUEGENERATOR (NAME: name, DRAWFUNCTION: drawFn), d,
    {fluents, defaults, object-generators, value-generators, fluent-generators, performance}) ≡
    {fluents, defaults, object-generators,
    {vg ∈ value-generators | value-generator-name(vg) ≠ value-generator-name(newGenerator)}
    ∪ {{name, drawFn}}, fluent-generators)

```

apply(ADDFLUENTGENERATOR (NAME: *name*, DRAWFUNCTION: *drawFn*), *d*,
 {*fluents*, *defaults*, *object-generators*, *value-generators*, *fluent-generators*, *performance*}) $\equiv$
 {*fluents*, *defaults*, *object-generators*, *value-generators*,
 {*fg* $\in$ *fluent-generators* | *fluent-generator-name*(*fg*) $\neq$ *fluent-generator-name*(*newGenerator*)}}
 $\cup$ {{*name*, *drawFn*}}}

apply(REPLACEPERFORMANCECALCULATION(PERFORMANCE: *calculation*), *d*,
 {*fluents*, *defaults*, *object-generators*, *value-generators*, *fluent-generators*, *performance*}) $\equiv$
 {*fluents*, *defaults*, *object-generators*, *value-generators*, *fluent-generators*, *calculation*}

3.5.3 Object Generator Draw Functions

The following simple example of an object generator draw function grounds our examples. Others, particularly with variable output, are possible.

OBJECTLIST(*count*, *id*) = () $\rightarrow$ [*gensym*(*id*, *i*)]_{*i*=1..*count*}

Generates *count* objects of the requested *type*, prepended with the string in *id*, as created by the Lisp “gensym” function.

3.5.4 Value Generators

The following examples of value generator draw functions ground our examples. Others are possible.

UNIFORMDISTRIBUTION(*min*, *max*)

Generator always returns a sequence containing a single real number drawn from a uniform distribution between min and max.

UNIFORMINTEGERDISTRIBUTION(*min*, *max*)

Generator always returns a sequence containing a single uniform integer draw between min and max (inclusive).

GAUSSIANDISTRIBUTION(*mean*, *stdev*)

Generator always returns a sequence containing a single draw from a one-dimensional Gaussian distribution with the given parameters.

NDIMENSIONALGAUSSIANDISTRIBUTION(*means*[], *stdevs*[])

Generator always returns a sequence containing a single draw from an n-dimensional Gaussian distribution with the given parameters, as a tuple with values named [x1, x2, ...].

BERNOULLIDISTRIBUTION(*p*)

Generator returns TRUE with probability *p*, and otherwise returns FALSE.

INTEGERSEQUENCE(*min*, *max*)

Generator always returns a sequence containing all integers between min and max, in order.

`DRAWFROMOBJECTSET(objectGeneratorName)`

Generator always returns a sequence containing a single draw from an object generator on the same scenario generator, referenced by name.

`DRAWALLFROMOBJECTSET(objectGeneratorName)`

Generator always returns a sequence containing all objects that were produced from an object generator on the same scenario generator, referenced by name, in a random order.

`CONSTANTFUNCTION(value)`

Generator always returns a sequence containing the single value passed in, i.e., the sequence [*value*].

`NDRAWS(valueGenerator, n)`

Generator returns a sequence of *n* values drawn from *valueGenerator*.

`DRAWTUPLE(valueGenerators[], names[])`

Generator returns a sequence with a single tuple with values drawn from *valueGenerators* and addressable by names given in *names*.

`FILTER(valueGenerator, filterFn (Any) → Boolean)`

Generator returns a sequence with a single value; this is the first value returned by repeatedly drawing from *valueGenerator* until the *filterFn* reports TRUE for one.

`NEWSET(valueGenerator)`

Generator always returns the same sequence, which is first populated by drawing from *valueGenerator*.

3.5.5 Fluent Generators

The following examples of fluent generator draw functions ground our examples. Others are possible.

`ALLPERMUTATIONS(argumentGenerators[], fluentValueGenerator)`

Generator returns ground fluents with arguments populated by value generators corresponding to the names in *argumentGenerator*, and values populated by value generators corresponding to *fluentValueGenerator* (can be referenced by name or provide draw function directly).

All permutations of a set of draws from the argument generators are used to generate a list of ground fluents. Values from *fluentValueGenerator* will be used in order, and a new set drawn when none are left, until every permutation is assigned a fluent value.

`NFLUENTDRAWS(argumentGenerators[], fluentValueGenerator, n)`

Generator returns *n* ground fluents. The arguments of each are populated by value generators corresponding to the names in *argumentGenerators*, and the value of each is populated by the value generator corresponding to *fluentValueGenerator*. All values from a single draw of each

argument and fluent value generator are used in order; if fewer than n values are returned by a generator, it will be called again until n values have been generated.

COMBINEFUNCTIONS(*fluentGenerators*[])

Results of each fluent generator are concatenated.

4 Application to Novelty Hierarchy

This section gives a novelty type definition for each level of the novelty hierarchy introduced by the DARPA SAIL-ON program (see Table 2). These definitions were an early attempt to categorize transformations possible in open-world environments, starting with the properties of individual world elements (objects, agents, and their actions and goals), advancing to multi-element properties (relations, interactions, and events), and ending with general properties broadly impacting multiple

Tier	Category	Definition	Notes
Single Entities	1 – Objects	An entity in the environment that is not the point-of-view agent and does not have goal-oriented behavior.	Objects may experience state changes as a result of actions by the i) point-of-view agent, ii) an external agent, iii) an event, iv) or a non-volitional behavior.
	2 – Agents	An entity in the environment that is not the point-of-view agent and has goal-oriented behavior. (People will disagree on what goal-oriented means, that is fine.)	Same as #1, except must have a goal. Goal does not have to be obvious. Random exploration can be a goal.
	3 – Actions	A goal-oriented behavior of an external agent that is not the point-of-view agent.	Goal does not have to be obvious. Random exploration can be a goal. Changes in movements and state-change attributes (a pendulum) of non-volitional entities (objects) are NOT considered actions.
Multiple Entities	4 – Relations	Static property of the relationship between multiple entities.	Can be spatial, temporal, or other. Can include any combination of entities (objects and agents) and can include the point-of-view agent.
	5 – Interactions	Dynamic property of behaviors or actions that impacts multiple entities.	Can be spatial, temporal, or other. Can include any combination of entities (objects and agents) and can include the point-of-view agent.
Complex Phenomena	6 – Environments	A change in an element of an open-world space that may impact the entire task space and is independent of a specific entity.	Can be general across the entire task space (e.g., global temperature change) or varied in space and time (e.g., partial or temporary darkness or wind).
	7 – Goals	A change in the purpose of goal-oriented behavior by an agent in the environment that is not the point-of-view agent.	If a domain already has team members of the AI agent (embodied or non-embodied) that alter goals, then changes in team or AI agent goal (directives) communicated by this external team member are in-scope.
	8 - Events	A state change or series of state changes that are not each the result of volitional action by an agent.	This includes state changes with specific preconditions.

Table 2. Novelty Hierarchy, ca. January 2022 (Kildebeck et al, 2022)

world elements (environments). Levels in the novelty hierarchy were intended to be mutually exclusive. Novelties of varying difficulty can arise from each level of the novelty hierarchy, though different novelty-handling techniques and more sophisticated world models may be required at higher numbered levels (Novelty Working Group, 2023). Note that the term “hierarchy” is used here for historical reasons; no hierarchical relationship is required or expected between these novelty characterizations.

The formalizations provided here show that the T-SAL R-Transformations and T-Transformations are sufficient for formalizing the novelty categorizations created for the SAIL-ON program. Future research should attempt to investigate what classes of novelty are difficult or easy for agents with various capabilities to reason about.

This section is organized as follows: there are eight novelty hierarchy levels. For each level, we first provide a natural language definition and clarifications shared in the DARPA SAIL-ON program (Kildebeck et al, 2022) to guide those groups responsible for generating novelty in these categories as to what novelties are admissible in each category. For each of the eight novelty hierarchy levels, we then provide a formal definition, first described in plain language then in L(T-SAL). The L(T-SAL) description of each level provides a decision function of the form *has*<X>*Novelty*(*t*, *d*), where “<X>” is the novelty type, *t* is a transformation sequence and *d* a T-SAL domain. This function decides whether the novelty described by *t* belongs to category X. Finally, for each level we provide one or more T-SAL-CR examples of the category as transformation sequences.

The ambiguity in the natural language definitions described below is such that novelties generated by various groups adhered to different standards (NWG, 2023). It is our hope that the formal standards presented here will help future researchers to adhere to common standards.

1 – Objects

Natural language definition: An entity in the environment that does not have goal-oriented behaviors.

Clarifications: Objects may experience state changes as a result of actions by i) the point-of-view agent, ii) an external agent, or iii) an event. This level is explicitly differentiated from “instance” novelty, which arises when new objects of an existing type are observed.

Formal definition: To introduce object novelty, a transformation sequence must either 1) add a new type that inherits from OBJECT, 2) add a new function that relates that object type to something else, or 3) change the initial distribution of such a function (via a new fluent generator). In addition, to ensure relevance, the new type or function must impact domain dynamics, through the preconditions of an action model, the conditions of a process model, and/or the triggers of an event model.

hasObjectNovelty (*t*, *d*, *sg*) $\equiv$
 $\exists i, f, j, p, c, d_N: 0 \leq i < |t|$
 $\wedge d_N = \text{applySequence}(t, d)$
 $\wedge (((\text{transformation-type}(t_i) = \text{ADDTYPE}$
 $\vee \text{transformation-type}(t_i) = \text{ADDTYPEPARENT}$
 $\vee \text{transformation-type}(t_i) = \text{REMOVETYPEPARENT})$
 $\wedge p = \text{transformation-argument}(t_i, \text{PARENT})$
 $\wedge c = \text{transformation-argument}(t_i, \text{CHILD})$
 $\wedge f \in \text{domain-functions}(d)$

$$\begin{aligned}
& \wedge \text{derived-from}(c, \text{OBJECT}, d_N) \\
& \wedge \text{derived-from}(p, \text{function-argument-types}(f)_i, d) \\
& \wedge \neg \text{derived-from}(c, \text{function-argument-types}(f)_j, d) \\
& \wedge \text{relevant-function}(\text{function-name}(f), d) \\
\vee & (\text{transformation-type}(\mathbf{t}_i) = \text{ADDFUNCTION} \wedge f = \text{transformation-argument}(\mathbf{t}_i, \text{FUNCTION}) \\
& \quad \wedge \text{derived-from}(\text{function-argument-types}(f)_i, \text{OBJECT}) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d_N)) \\
\vee & (\text{transformation-type}(\mathbf{t}_i) = \text{ADDFLUENTGENERATOR} \\
& \quad \wedge f = \text{fluent-generator-function}(\text{transformation-argument}(\mathbf{t}_i, \text{FLUENTGENERATOR})) \\
& \quad \wedge \text{derived-from}(\text{function-argument-types}(f)_i, \text{OBJECT}) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d_N))
\end{aligned}$$

Required Definitions:

The relation *derived-from* is defined in Appendix A.

relevant-function(fn, d) $\equiv$

$$\begin{aligned}
& (\exists \text{cnd}: ((\exists a \in \text{domain-actions}(d) \wedge \text{cnd} \in \text{action-preconditions}(a)) \\
& \quad \vee (\exists \text{ev} \in \text{domain-events}(d) \wedge \text{cnd} \in \text{event-triggers}(\text{ev})) \vee \\
& \quad \vee (\exists p \in \text{domain-processes}(d) \wedge \text{cnd} \in \text{process-conditions}(\text{ev})) \\
& \quad \wedge (\text{function-in-condition}(\text{fn}, \text{cnd})) \\
& \quad \vee \exists ax: \text{function-affects-axiom}(\text{fn}, ax, d) \wedge \text{function-in-condition}(\text{fn}, \text{cnd})) \\
& \vee \text{function-affects-performance}(\text{fn}, d)
\end{aligned}$$

function-affects-performance(fn, d) $\equiv$

$$\text{function-in-calculation}(\text{fn}, \text{domain-performance-function}(d))$$

function-affects-axiom(fn, ax, d) $\equiv$

$$\begin{aligned}
& \text{function-in-condition}(\text{fn}, \text{axiom-condition}(ax), d) \\
& \vee \exists ax_M: ax_M \in \text{domain-axioms}(d) \\
& \quad \wedge \text{function-affects-axiom}(\text{fn}, ax_M, d) \\
& \quad \wedge \text{function-in-condition}(\text{axiom-name}(ax_M), \text{axiom-condition}(ax), d)
\end{aligned}$$

function-in-term(fn, te $\in Te$) $\equiv \perp$

function-in-term(fn, te = $\langle f, \mathbf{te} \rangle \in F \times \mathbf{Te}$) $\equiv$

$$\begin{aligned}
& \text{function-name}(f) = \text{fn} \\
& \vee (\exists i: 0 \leq i < |\mathbf{te}| \wedge \text{function-in-calculation}(\text{fn}, \mathbf{te}_i))
\end{aligned}$$

function-in-condition(fn, cnd $\in Te$) $\equiv$

$$\text{function-in-term}(\text{fn}, \text{cnd})$$

function-in-condition(fn, $\langle \text{bop}, \text{cnd1}, \text{cnd2} \rangle \in \text{bOp} \times \text{Cn} \times \text{Cn}$) $\equiv$

$$\begin{aligned}
& \text{function-in-condition}(\text{fn}, \text{cnd1}) \\
& \vee \text{function-in-condition}(\text{fn}, \text{cnd2})
\end{aligned}$$

function-in-condition(fn, $\langle \text{FORALL}, \mathbf{v}, \text{cnd1}, \text{cnd2} \rangle$) $\equiv$

$$\begin{aligned}
& \text{function-in-condition}(\text{fn}, \text{cnd1}) \\
& \vee \text{function-in-condition}(\text{fn}, \text{cnd2})
\end{aligned}$$

$$\begin{aligned}
& \text{function-in-condition}(fn, \langle ineq, calc1, calc2 \rangle \in Ineq \times Ca \times Ca) \equiv \\
& \quad \text{function-in-calculation}(fn, calc1) \\
& \quad \vee \text{function-in-calculation}(fn, calc2) \\
& \text{function-in-calculation}(fn, calc \in Te) \equiv \\
& \quad \text{function-in-term}(fn, calc) \\
& \text{function-in-calculation}(fn, \langle op, calc1, calc2 \rangle \in Op \times Ca \times Ca) \equiv \\
& \quad \text{function-in-calculation}(fn, calc1) \vee \text{function-in-calculation}(fn, calc2) \\
& \text{function-in-calculation}(fn, \langle aggop, v, con, calc \rangle \in AggOp \times V \times Cn \times Ca) \equiv \\
& \quad \text{function-in-condition}(fn, con) \vee \text{function-in-calculation}(fn, calc) \\
& \text{function-in-calculation}(fn, \langle if, cnd, calc1, calc2 \rangle \in IF \times Cn \times Ca \times Ca) \equiv \\
& \quad \text{function-in-condition}(fn, cnd) \vee \text{function-in-calculation}(fn, calc1) \vee \text{function-in-calculation}(fn, calc2) \\
& \text{function-argument-types}(f) = [\text{function-argument-type}(f, \text{function-argument}(f)_i)]_{i=1..|\text{function-argument}(f)|} \\
& \text{fluent-generator-function}(fg, d) = \text{argmin}_f f \in \text{domain-functions}(d) \\
& \quad \wedge \text{function-name}(f) = \text{fluent-generator-function-name}(fg)
\end{aligned}$$

Example: Every cart starts with a block within 2 distance units (3 carts standard):

```

ADDVALUEGENERATOR(NEARBY-STARTS, FILTER(
    DRAWTUPLE([POSITION, POSITION], ["cart", "block"])),
    λ(x, y) → {|x - y| < 2})
ADDVALUEGENERATOR(START-POSITIONS, NEWSET(NDRAWS(NEARBY-STARTS, 3))
ADDVALUEGENERATOR(CLOSE-BLOCKS, NDRAWS(BLOCKGROUP, 3))
ADDVALUEGENERATOR(FAR-BLOCKS, DIFFERENCE(BLOCKGROUP, CLOSE-BLOCKS))
ADDFLUENTGENERATOR(CART-POSITION, ALLPERMUTATIONS([CARTGROUP], START-POSITION.CART)
ADDFLUENTGENERATOR(BLOCK-POSITION,
    COMBINEFUNCTIONS(ALLPERMUTATIONS([CLOSE-BLOCKS], START-POSITION.BLOCK)
        ALLPERMUTATIONS([FAR-BLOCKS], POSITION))

```

Example: Initial block velocities are doubled:

Original generator:

```

FLUENTGENERATOR(BLOCK-VELOCITY, ALLPERMUTATIONS([BLOCKGROUP], RANDOMINSET([1, -1])
ADDFLUENTGENERATOR(BLOCK-VELOCITY, ALLPERMUTATIONS([BLOCKGROUP], RANDOMINSET([2, -2])

```

2 – Agents

Natural language definition: An entity in the environment that is not the point-of-view agent and has goal-oriented behavior. The agent is most frequently an external agent but can in some instances be the point-of-view agent. The goal of the agent does not have to be obvious.

Clarifications: Researchers will disagree on what ‘goal-oriented behavior’ means. This is fine, and researchers have flexibility within reason to determine best practices in specific domains. The goal of the agent does not have to be obvious. For example, random exploration could be a goal.

Formal definition: An agent-level novelty is defined identically to an object-level novelty, but using AGENT subtypes instead of OBJECT subtypes. Note that agents are primarily distinguished from objects in that every action has a performer which must be an agent.

$$\begin{aligned}
& \text{hasAgentNovelty}(\mathbf{t}, d) \equiv \\
& \exists i, f, j, p, c, d_N: 0 \leq i < |\mathbf{t}| \\
& \quad \wedge d_N = \text{applySequence}(\mathbf{t}, d) \\
& \quad \wedge (((\text{transformation-type}(\mathbf{t}_i) = \text{ADDTYPE} \\
& \quad \quad \vee \text{transformation-type}(\mathbf{t}_i) = \text{ADDTYPEPARENT} \\
& \quad \quad \vee \text{transformation-type}(\mathbf{t}_i) = \text{REMOVETYPEPARENT}) \\
& \quad \wedge p = \text{transformation-argument}(\mathbf{t}_i, \text{PARENT}) \\
& \quad \wedge c = \text{transformation-argument}(\mathbf{t}_i, \text{CHILD}) \\
& \quad \wedge f \in \text{domain-functions}(d) \\
& \quad \wedge \text{derived-from}(c, \text{AGENT}, d_N) \\
& \quad \wedge \text{derived-from}(p, \text{function-argument-types}(f)_j, d) \\
& \quad \wedge \neg \text{derived-from}(c, \text{function-argument-types}(f)_j, d) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d)) \\
& \vee (\text{transformation-type}(\mathbf{t}_i) = \text{ADDFUNCTION} \wedge f = \text{transformation-argument}(\mathbf{t}_i, \text{FUNCTION}) \\
& \quad \wedge \text{derived-from}(\text{function-argument-types}(f)_j, \text{AGENT}) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d_N)) \\
& \vee (\text{transformation-type}(\mathbf{t}_i) = \text{ADDFLUENTGENERATOR} \\
& \quad \wedge f = \text{fluent-generator-function}(\text{transformation-argument}(\mathbf{t}_i, \text{FLUENTGENERATOR})) \\
& \quad \wedge \text{derived-from}(\text{function-argument-types}(f)_j, \text{AGENT}) \\
& \quad \wedge \text{relevant-function}(\text{function-name}(f), d_N))
\end{aligned}$$

Required Definitions:

The relation *derived-from* is defined in Appendix A.

Example: A transformation sequence changes the amount of force other agents use when pushing, referenced by the push action (see Section 3.1).

$$\begin{aligned}
& \text{ADDVALUEGENERATOR}(\text{OTHER-AGENTS}, \text{NEWSET}(\text{DIFFERENCE}(\text{AGENTS}, [\text{AGENT1}]))) \\
& \text{ADDFLUENTGENERATOR}(\text{AGENT-FORCE}, \text{ALLPERMUTATIONS}([\text{OTHER-AGENTS}], \\
& \quad \text{UNIFORMDISTRIBUTION}(0.5, 1.5)))
\end{aligned}$$

3 – Actions

Natural language definition: A goal-oriented behavior of an external agent that is not the point-of-view agent.

Clarifications: Non-goal-oriented movements of non-volitional objects are not considered actions.

Formal definition: For a transformation sequence to exhibit action novelty, the preconditions or effects of an action model must be changed. In addition, we require that the performer of the changed action cannot be of the same type ty_{AG} as the point-of-view agent.

$$\begin{aligned}
& \text{hasActionNovelty}(\mathbf{t} = [\mathbf{t}_0 \dots \mathbf{t}_n], ty_{AG}, d) = \\
& \exists i < |\mathbf{t}|, d_p, d_T, a_T, n, v_T, \theta, \mathbf{v}_{BOUND}: \\
& \quad d_p = \text{applySequence}([\mathbf{t}_0 \dots \mathbf{t}_{i-1}], d) \\
& \quad \wedge \exists a \in \text{domain-actions}(d_p): n = \text{action-name}(a) \\
& \quad \wedge \text{transformation-argument}(\mathbf{t}_i, \text{ACTIONNAME}) = n
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{transformation-type}(\mathbf{t}_i) \in \{\text{ADDPRECONDITION}, \text{REMOVEPRECONDITION}, \\
& \quad \text{ADDACTIONEFFECT}, \text{REMOVEACTIONEFFECT}\} \\
& \wedge d_T = \text{applySequence}(\mathbf{t}, d) \\
& \wedge a_T \in \text{domain-actions}(d_T) \wedge \text{action-name}(a_T) = n \\
& \wedge \text{action-performer}(a_T) = v_T \\
& \wedge \mathbf{v}_{\text{BOUND}} = \text{bound-variables-in-conditions}(\text{action-preconditions}(a_T), \mathbf{v}_{\text{BOUND}}) \\
& \quad \cup (\{\text{action-performer}(a_T)\} \cap V) \cup (\text{action-parameters}(a_T)) \\
& \wedge \neg \exists \theta: (\text{assignable-to}(\text{ty}_{AG}, v_T, \theta, d_T) \\
& \quad \wedge \forall \text{cnd} \in \text{action-preconditions}(a_T): \text{legal-condition}(\text{cnd}, \theta, \mathbf{v}_{\text{BOUND}}, d))
\end{aligned}$$

Required Definitions:

The relation *bound-variables-in-conditions* and *assignable-to* are defined in Appendix A.

Example: An environment transformation adds an action that lets agents “donate” cart velocity to blocks

```

ADDACTION(DONATE, ?AG, [?C], [CART])
ADDPRECONDITION(DONATE, (= (CONTROLS ?AG ?C) TRUE))
ADDPRECONDITION(DONATE, (= (CART-VELOCITY ?C) ?V))
ADDACTIONEFFECT(DONATE, (SET (CART-VELOCITY ?C) 0))
ADDACTIONEFFECT (DONATE, (INCREASE (BLOCK-VELOCITY ?B) ?V))

```

Example: Cart pushes have a budget

```

ADDFUNCTION(PUSH-BUDGET)
ADDPRECONDITION(PUSH, (> (PUSH-BUDGET ?AG) ?FORCE))
ADDACTIONEFFECT (PUSH, (DECREASE (PUSH-BUDGET ?AG) ?FORCE))
ADDFLUENTGENERATOR(PUSH-BUDGET, ALLPERMUTATIONS([AGENTS], CONSTANTFUNCTION(100)))

```

4 – Relations

Natural language definition: Static properties of the relationships between multiple entities.

Clarifications: Can be spatial, temporal, or other. Can include the point-of-view agent and other entities.

Formal definition: A transformation sequence exhibits relation novelty if a static function (one not present in any effects or process changes) is created, has its distribution changed, or is added to triggers, conditions, or preconditions. The function must be relevant and must involve 2 or more entities (here, entities are specifically objects of types AGENT, OBJECT, or their subtypes).

hasRelationNovelty($\mathbf{t}, d$) =

$$\begin{aligned}
& \exists f, \text{name}, i: 0 \leq i < |\mathbf{t}| \\
& \wedge ((\text{transformation-type}(\mathbf{t}_i) = \text{ADDFUNCTION} \\
& \quad \wedge \text{function-name}(\text{transformation-argument}(\mathbf{t}_i, \text{FUNCTION})) = \text{name}) \\
& \vee (\text{transformation-type}(\mathbf{t}_i) \in \{\text{ADDPRECONDITION}, \text{REMOVEPRECONDITION}\} \\
& \quad \wedge \text{function-in-condition}(\text{name}, \text{transformation-argument}(\mathbf{t}_i, \text{PRECONDITION}))) \\
& \vee (\text{transformation-type}(\mathbf{t}_i) \in \{\text{ADDTRIGGER}, \text{REMOVETRIGGER}\} \\
& \quad \wedge \text{function-in-condition}(\text{name}, \text{transformation-argument}(\mathbf{t}_i, \text{TRIGGER}))) \\
& \vee (\text{transformation-type}(\mathbf{t}_i) \in \{\text{ADDPROCESSCONDITION}, \text{REMOVEPROCESSCONDITION}\}
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{function-in-condition}(\text{name}, \text{transformation-argument}(\mathbf{t}_i, \text{CONDITION}))) \\
& \vee (\text{transformation-type}(\mathbf{t}_i) = \text{ADDFLUENTGENERATOR} \\
& \quad \wedge \text{name} = \text{fluent-generator-function-name}(\text{transformation-argument}(\mathbf{t}_i, \text{FLUENTGENERATOR}))) \\
& \wedge d_T = \text{applySequence}(\mathbf{t}, d) \\
& \wedge f \in \text{domain-functions}(d_T) \\
& \wedge \text{function-name}(f) = \text{name} \\
& \wedge |\text{entities-in-function}(f)| > 1 \\
& \wedge (\forall a \in \text{domain-actions}(d_T) : \nexists e \in \text{action-effects}(a) : \text{effect-name}(e) = \text{name} \\
& \quad \vee \forall ev \in \text{domain-events}(d_T) : \nexists e \in \text{event-effects}(ev) : \text{effect-name}(e) = \text{name} \\
& \quad \vee \forall p \in \text{domain-processes}(d_T) : \nexists c \in \text{process-changes}(p) : \text{change-name}(c) = \text{name})
\end{aligned}$$

Required Definitions:

$$\begin{aligned}
& \text{entities-in-function}(f) = \\
& \quad \{v \in \text{function-arguments}(f) \mid \text{derived-from}(\text{function-argument-type}(f,v), \text{OBJECT}) \\
& \quad \vee \text{derived-from}(\text{function-argument-type}(f,v), \text{AGENT})\}
\end{aligned}$$

Example: A transformation sequence asserts a minimum distance between every block and cart. This adds a new binary static function, MIN-DIST. The revised scenario generator assigns a random value between 0 and 1 to this function for each pair of block and cart. The PUSH action preconditions change to enforce this constraint.

$$\begin{aligned}
& \text{ADDFUNCTION}(\langle \text{MIN-DIST}, [\text{?B}, \text{?C}], [\text{BLOCK}, \text{CART}], \text{REAL} \rangle) \\
& \text{ADDPRECONDITION}(\text{PUSH}, (= (\text{CART-POSITION ?C}) ?P)) \\
& \text{ADDPRECONDITION}(\text{PUSH}, (\text{FORALL ?B (IS-BLOCK ?B)} \\
& \quad (\text{AND} (< (- (+ ?P ?FORCE) (\text{BLOCK-POSITION ?B})) (\text{MIN-DIST ?B ?C})) \\
& \quad (< (- (\text{BLOCK-POSITION ?B}) (+ ?P ?FORCE)) (\text{MIN-DIST ?B ?C})))))) \\
& \text{ADDFLUENTGENERATOR}(\text{MIN-DIST}, \text{ALLPERMUTATIONS}([\text{BLOCKS}, \text{CARTS}], \text{UNIFORMDISTRIBUTION}(0, 1)))
\end{aligned}$$

5 – Interactions

Natural language definition: Dynamic property of behaviors or actions that impacts multiple entities.

Clarifications: Can be spatial, temporal, or other. Can include the point-of-view agent and other entities.

Formal definition: A transformation sequence exhibits interaction novelty if a fluent is added to or changed in the effects of an action. The fluent must name a relevant function that involves 2 or more entities (here, entities are specifically objects of types AGENT, OBJECT, or their subtypes).

$$\begin{aligned}
& \text{hasInteractionNovelty}(\mathbf{t}, d) = \\
& \quad \exists f, \text{fname}, \text{aname}, i: 0 \leq i < |\mathbf{t}| \\
& \quad \wedge ((\text{transformation-type}(\mathbf{t}_i) \in \{\text{ADD ACTION EFFECT}, \text{REMOVE ACTION EFFECT}\} \\
& \quad \quad \wedge \text{effect-name}(\text{transformation-argument}(\mathbf{t}_i, \text{EFFECT})) = \text{fname}) \\
& \quad \wedge \exists a \in \text{domain-actions}(d): \\
& \quad \quad \text{aname} = \text{action-name}(a) \\
& \quad \quad \wedge \text{transformation-argument}(\mathbf{t}_i, \text{ACTIONNAME}) = \text{aname}
\end{aligned}$$

$$\begin{aligned}
&\wedge \text{function-name}(f) = \text{fname} \\
&\wedge |\text{entities-in-function}(f)| > 1 \\
&\wedge d_\tau = \text{applySequence}(\mathbf{t}, d) \\
&\wedge (\exists a_\tau \in \text{domain-actions}(d_\tau) : \\
&\quad \text{action-name}(a_\tau) = \text{aname} \\
&\quad \wedge \exists e \in \text{action-effects}(a_\tau) : \text{effect-name}(e) = \text{fname})
\end{aligned}$$

Example: A new action allows an agent to move to a nearby cart; this affects the binary function CONTROLS, which becomes dynamic and affects the relationship between agent and cart.

```

ADDACTION(SWITCH-CARTS, [?C1, ?C2], [CART, CART])
ADDPRECONDITION(SWITCH-CARTS, (= (CONTROLS ?AG ?C1) TRUE))
ADDPRECONDITION(SWITCH-CARTS, (= (CONTROLS ?AG ?C2) FALSE))
ADDPRECONDITION(SWITCH-CARTS, (= (CART-POSITION ?C1) ?P1))
ADDPRECONDITION(SWITCH-CARTS, (= (CART-POSITION ?C2) ?P2))
ADDPRECONDITION(SWITCH-CARTS, (< (- ?P2 ?P1) 1))
ADDPRECONDITION(SWITCH-CARTS, (< (- ?P1 ?P2) 1))
ADDACTIONEFFECT(SWITCH-CARTS, (= (CONTROLS ?AG ?C2) TRUE))
ADDACTIONEFFECT(SWITCH-CARTS, (= (CONTROLS ?AG ?C1) FALSE))

```

6 – Environment

Natural language definition: A change in an element of an open-world space that may impact the entire task space and is independent of a specific entity.

Clarifications: Environment novelties may impact the entire task space identically (e.g., changing the temperature from a static value to a new static value) or may impact regions of the open world domain differentially (e.g., wind may be present in only part of a space and the intensity may ebb and flow over time).

Formal definition: An environment novelty is a transformation sequence in which environmental functions, processes, or events are modified. Environmental functions are those functions outside of agent control, meaning they are not affected by functions. Environmental processes and events are those that are conditioned on or triggered by only environmental functions and position-valued functions.

$$\begin{aligned}
&\text{hasEnvironmentNovelty}(\mathbf{t}, d) = \\
&\quad (\exists i, d_\tau: \\
&\quad \quad 0 \leq i < |\mathbf{t}| \\
&\quad \quad \wedge d_\tau = \text{applySequence}(\mathbf{t}, d) \\
&\quad \quad \wedge (\text{transformation-type}(\mathbf{t}_i) = \text{ADDFLUENTGENERATOR} \\
&\quad \quad \quad \wedge \exists f, \text{fname}: \\
&\quad \quad \quad \quad \text{isEnvironmentalFunction}(f, d_\tau) \\
&\quad \quad \quad \quad \wedge \text{fname} = \text{function-name}(f) \\
&\quad \quad \quad \quad \wedge \text{transformation-argument}(\mathbf{t}_i, \text{NAME}) = \text{fname}) \\
&\quad \quad \vee (\text{transformation-type}(\mathbf{t}_i) \in \{\text{ADDTRIGGER}, \text{REMOVETRIGGER}, \text{ADDEVENTEFFECT}, \text{REMOVEVENTEFFECT}, \\
&\quad \quad \quad \text{CHANGEPROBABILITY}, \text{CHANGEFREQUENCY}\})
\end{aligned}$$

$$\begin{aligned}
& \wedge \exists ev \in \text{domain-events}(d_T): \\
& \quad \text{event-name}(ev) = \text{transformation-argument}(t_i, \text{EVENTNAME}) \\
& \quad \wedge \text{isEnvironmentalEvent}(ev, d_T) \\
& \forall (\text{transformation-type}(t_i) \in \{\text{ADDPROCESSCONDITION}, \text{REMOVEPROCESSCONDITION}, \\
& \quad \text{ADDPROCESSCHANGE}, \text{REMOVEPROCESSCHANGE}\}) \\
& \wedge \exists p \in \text{domain-processes}(d_T): \\
& \quad \text{process-name}(p) = \text{transformation-argument}(t_i, \text{PROCESSNAME}) \\
& \quad \wedge \text{isEnvironmentalProcess}(p, d_T)
\end{aligned}$$

Required definitions:

$$\begin{aligned}
& \text{isEnvironmentalFunction}(f, d) = \\
& \quad f \in \text{domain-functions}(d) \\
& \quad \wedge \neg(\exists a, e: a \in \text{domain-actions}(d) \wedge e \in \text{action-effects}(a) \wedge \text{effect-name}(e) = \text{function-name}(f) \\
& \quad \wedge |\text{entities-in-function}(f)| = 0)
\end{aligned}$$

$$\begin{aligned}
& \text{isEnvironmentalProcess}(p, d) = \\
& \quad \exists cnd \in \text{process-preconditions}(ev), f \in \text{domain-functions}(d): \\
& \quad \quad \text{function-in-condition}(\text{function-name}(f), cnd) \wedge \text{isEnvironmentalFunction}(d, f) \\
& \quad \forall cnd \in \text{process-preconditions}(p), f \in \text{domain-functions}(d): \\
& \quad \quad \neg \text{function-in-condition}(\text{function-name}(f), cnd) \\
& \quad \quad \vee \text{isEnvironmentalFunction}(d, f) \vee \text{function-value-type}(f) = \text{POSITION}
\end{aligned}$$

$$\begin{aligned}
& \text{isEnvironmentalEvent}(ev, d) = \\
& \quad \exists cnd \in \text{event-triggers}(ev), f \in \text{domain-functions}(d): \\
& \quad \quad \text{function-in-condition}(\text{function-name}(f), cnd) \wedge \text{isEnvironmentalFunction}(d, f) \\
& \quad \forall cnd \in \text{event-triggers}(ev), f \in \text{domain-functions}(d): \\
& \quad \quad \neg \text{function-in-condition}(\text{function-name}(f), cnd) \\
& \quad \quad \vee \text{isEnvironmentalFunction}(d, f) \vee \text{function-value-type}(f) = \text{POSITION}
\end{aligned}$$

Example: A transformation introduces a “gravitational jump” that distorts pole angles every 20 time units. This is modeled by an environmental event, JUMP-PULLS.

```

ADDFUNCTION(JUMP-TIME, [], [])
ADDPROCESS(JUMP-TICKS, [], [])
ADDPROCESSCHANGE(JUMP-TICKS, (INCREASE (JUMP-TIME) (* DT 1)))
ADDEVENT(JUMP-PULLS, [?C], [CART])
ADDTRIGGER(JUMP-PULLS, (> (JUMP-TIME) 20))
ADDEVENTEFFECT(JUMP-PULLS, (INCREASE (POLE-ANGLE ?C) 5))
ADDEVENTEFFECT(JUMP-PULLS, (SET (JUMP-TIME) 0))
ADDFLUENTGENERATOR(JUMP-TIME, ALLPERMUTATIONS([], UNIFORMRANDOM(0, 20)))

```

7 – Goals

Natural language definition: The purpose of a behavior by an agent in the environment.

Clarifications: Goal novelties resulting from environmental transformations are primarily considered to be goal changes of external agents in the environment that are not the point-of-view agent. Some domains may include teammates where the task goal is dynamic and communicated to the point-of-view agent as a directive or change in observable reward. In such domains, changes in the goals of the team or point-of-view agent communicated by external team members can be considered in-scope, but not required for point-of-view.

Formal definition: Goal novelties are those that include a T-Transformation changing how the performance function of a scenario generator is calculated.

$hasGoalNovelty(t, d) \equiv$

$\exists t \in \mathbf{t}: transformation\text{-}type(t) = REPLACEPERFORMANCECALCULATION$

Example: An environment transformation introduces a new performance calculation that rewards an agent lasting without winning or losing.

```
ADDFUNCTION(CLOCK-TIME, [], [])
ADDPROCESS(CLOCK-TICKS, [], [])
ADDPROCESSCHANGE(CLOCK-TICKS, (INCREASE (CLOCK-TIME) (* DT 1)))
REPLACEPERFORMANCECALCULATION((CLOCK-TIME))
ADDDFAULT(CLOCK-TIME, 0)
```

8 – Events

Natural language definition: A state change or series of state changes that are not the result of volitional action by an external agent or the point-of-view agent.

Clarifications: Events include state changes with specific preconditions.

Formal definition: An event novelty is a transformation sequence that affects T-SAL events that have both environmental and non-environmental triggers. At least one trigger of a modified event must be based on an environment function and at least one must be directly affected by an action. Event novelty sequences include those that modify the distribution of an environmental function that triggers such an event.

$hasEventNovelty(t, d) \equiv$

$\exists ev, d_T, i:$

$ev \in domain\text{-}events(d_T)$

$\wedge d_T = applySequence(\mathbf{t}, d)$

$\wedge \neg isEnvironmentalEvent(ev, d_T)$

$\wedge 0 \leq i < |\mathbf{t}|$

$\wedge ((transformation\text{-}type(\mathbf{t}_i) \in \{ADDTRIGGER, REMOVETRIGGER, ADDEVENTEFFECT, REMOVEEVENTEFFECT, \\ CHANGEPROBABILITY, CHANGEFREQUENCY\})$

$\wedge event\text{-}name(ev) = transformation\text{-}argument(\mathbf{t}_i, EVENTNAME))$

$\vee (transformation\text{-}type(\mathbf{t}_i) = ADDFLUENTGENERATOR$

$\wedge \exists f: isEnvironmentalFunction(f, d_T)$

$$\wedge \text{transformation-type}(t, \text{NAME}) = \text{function-name}(f)$$

$$\wedge \exists \text{cnd} \in \text{event-triggers}(\text{ev}): \text{function-in-condition}(\text{function-name}(f), \text{cnd}))$$

Example: An environment transformation introduces a “gravitational anomaly” that pulls nearby carts and blocks based on their position, if they are moving. This is modelled by a non-environmental event, GRAVITY-PULLS, with a varying position trigger, an environmental function trigger, and a third trigger (velocity) under the agent’s control.

```

ADDFUNCTION(GRAVITY-LOCATION)
ADDPROCESS(GRAVITY-WAVE-MOVES, [], [])
ADDPROCESSCHANGE(GRAVITY-WAVE-MOVES, (INCREASE (GRAVITY-LOCATION) (* DT 1)))
ADDEVENT(GRAVITY-WAVE-RESETS, [], [])
ADDTRIGGER(GRAVITY-WAVE-RESETS, (>= (GRAVITY-LOCATION) 20))
ADDEVENTEFFECT(GRAVITY-WAVE-RESETS, (SET (GRAVITY-LOCATION) -20))
ADDEVENT(GRAVITY-PULLS, [?C], [CART])
CHANGEEVENTFREQUENCY(GRAVITY-PULLS, 4)
ADDTRIGGER(GRAVITY-PULLS, (> (CART-VELOCITY ?C) .01)))
ADDTRIGGER(GRAVITY-PULLS, (= (CART-POSITION ?C) ?P)))
ADDTRIGGER(GRAVITY-PULLS, (= (GRAVITY-LOCATION) ?WAVE-POS)))
ADDEVENTEFFECT(GRAVITY-PULLS, (INCREASE (CART-VELOCITY ?C) (* DT (LOG-DISTANCE ?P ?WAVE-POS))))
ADDFLUENTGENERATOR(GRAVITY-LOCATION, ALLPERMUTATIONS([], UNIFORMRANDOM(-20, 20)))

```

5 Discussion and Future Work

In this work, we have set up and formally defined a framework for describing a general class of environments and environment transformations. This is a necessary precursor to general agreement on definitions and measurement of the robustness of artificially intelligent agents. These definitions will be useful to future studies of general agents that take on very difficult situations flexibly and adaptively without human assistance. The “novelty hierarchy” is a first step in exploring possible classes of transformation sequences. Future work will use the formalism described here to define more useful assumptions to guide research.

Human design choices about T-SAL environments may have significant impact on how a transformation is described. In some cases, this has been observed to change category membership for the novelty hierarchy described here. This ambiguity is exacerbated when considering agents that have learned their own environment descriptions. To such an agent, the original environment may have a different definition than to a describing human – even if it’s consistent with the same observations. Therefore, a novelty hierarchy categorization might be different than to an observing human.

A known failing of T-SAL-CR is that it does not describe observation of the environment; clearly, the environment is responsible for providing an agent affordances in observation, which can change over time. For example, a robot may gain or lose awareness of its location via a GPS sensor. T-SAL-CR cannot describe what aspects of the state are observable, and therefore the T-SAL transformation language also cannot describe changes in observability. Future work should remedy this problem.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001121C0236. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

References

- Balloch, J., Lin, Z., Hussain, M., Srinivas, A., Wright, R., Peng, X., Kim, J., & Riedl, M. (2022). NovGrid: A Flexible Grid World for Evaluating Agent Response to Novelty. *arXiv preprint arXiv:2203.12117*.
- Boult, T. E., Grabowicz, P. A., Prijatelj, D. S., Stern, R., Holder, L., Alspector, J., ... & Scheirer, W. J. (2021). Towards a Unifying Framework for Formal Theories of Novelty. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 17, pp. 15047-15052).
- Edelkamp, S., & Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. University of Freiburg.
- Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-208.
- Fox, M., & Long, D. (2003). pddl2.1 : An Extension to pddl for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20, 61-124.
- Fox, M., & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27, 235-297.
- Gamage, C., Pinto, V., Xue, C., Stephenson, M., Zhang, P., & Renz, J. (2021). Novelty Generation Framework for AI Agents in Angry Birds Style Physics Games. *IEEE Conference on Games (IEEE-COG'21)*. Copenhagen, Denmark.
- Gerevini, A., & Long, D. (2005). Plan Constraints and Preferences in PDDL3 The Language of the Fifth International Planning Competition 1 Motivations and Goals. Brescia, Italy.
- Goel, S., Tatiya, G., Scheutz, M., & Sinapov, J. (2021). Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds. *International Foundation for Autonomous Agents and Multiagent Systems, AAMAS*.
- Helmert, M. (2008). Changes in PDDL 3.1. Retrieved from <http://icapsconference.org/ipc2008/deterministic/PddlExtension.html>
- Kejriwal, M., & Thomas, S. (2021). A multi-agent simulator for generating novelty in monopoly. *Simulation Modelling Practice and Theory*, 112, 102364.
- Kildebeck, E., Holder, L., Kejriwal, L., Renz, J., Robertson, E., & Shrivastava, A. (2022). *2022 Novelty Hierarchy*. Unpublished manuscript.
- Novelty Working Group. (2023). *Working Sessions of the Novelty Working Group* [Weekly Meetings]. Virtual Sessions.
- Langley, P. (2020). Open-world learning for radically autonomous agents. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 09, pp. 13539-13543).
- McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., . . . Wilkins, D. (1998). PDDL | The Planning Domain Definition Language. Yale University, New Haven, CT.
- Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.
- Molineaux, M., & Aha, D. (2014, June). Learning unknown event models. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 28, No. 1).
- Molineaux, M. & Dannenhauer, D. (2021). An Environment Transformation-based Framework for Comparison of Open-World Learning Agents. AAAI Spring Symposium: Designing Artificial Intelligence for Open Worlds. Virtual.
- Muhammad, F., Sarathy, V., Tatiya, G., Goel, S., Gyawali, S., Guaman, M., Sinapov, J., & Scheutz, M. (2021). A Novelty-Centric Agent Architecture for Changing Worlds. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*.
- Orwant, J. (2000). EGGG: Automated programming for game generation. *IBM Systems Journal*, vol. 39, no. 3-4, pp. 782-794.
- Pell, B. (1992). METAGAME: A new challenge for games and learning. in *Heuristic Programming in Artificial Intelligence 3*, H. Van den Kerik and L. Allis, Eds. Chichester, U.K.: Ellis Horwood.
- Smith, A. M., & Mateas, M. (2010). Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games* (pp. 273-280). IEEE.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7), 449-458.
- Xue, C., Pinto, V., Zhang, P., Gamage, C., Nikonova, E., & Renz, J. (2022). Science Birds Novelty: an Open-world Learning Test-bed for Physics Domains.

Appendix A T-SAL Environment Legality

The definitions below describe conditions for a legal T-SAL environment.

The relation *derived-from*: $Ty \times Ty \times D$ indicates whether two types have a derivation relationship; i.e., there is a path of supertypes in the domain from the proposed descendant type to the ancestor type. All top-level types that do not derive from other basic types derive from object.

$$\begin{aligned} \text{derived-from}(ty_D, ty_A, d) \equiv \\ & \langle ty_D, ty_A \rangle \in \text{domain-supertypes}(d) \\ & \forall ty: \langle ty_D, ty_A \rangle \in \text{domain-supertypes}(d) \wedge \text{derived-from}(ty, ty_A, d) \end{aligned}$$

$$\begin{aligned} \text{derived-from}(ty_D, \text{OBJECT}, d) \equiv \\ & \forall ty \in \text{domain-types}(d): \langle ty_D, ty \rangle \notin \text{domain-supertypes}(d) \\ & \wedge ty_D \notin \{\text{OBJECT}, \text{REAL}, \text{INTEGER}, \text{BOOLEAN}, \text{AGENT}\} \end{aligned}$$

To describe variable-type relationships, we use sets of $\langle \text{variable}, \text{type} \rangle$ pairs that allow mapping of a variables to an assigned type. We denote such a set $\theta = (\langle v_0, ty_0 \rangle, \langle v_1, ty_1 \rangle, \dots \langle v_n, ty_n \rangle)$. The function *variable-type*: $\theta \times V \rightarrow (S \cup \{\perp\})$ gives the type of a variable found in or $\perp$ if a variable is not found in the set; *variable-undefined* is true iff a variable is not Found in the set.

$$\begin{aligned} \text{variable-type}(\theta = (\langle v_0, ty_0 \rangle, \langle v_1, ty_1 \rangle, \dots \langle v_n, ty_n \rangle), v) = \\ \text{if } (|\theta| = 0, \text{TRUE: } \perp, \\ \text{FALSE: if } (v_0 = v, \text{TRUE: } ty_0, \\ \text{FALSE: variable-type}((\langle v_1, ty_1 \rangle, \dots \langle v_n, ty_n \rangle), v) \end{aligned}$$

$$\text{variable-undefined}(\theta = (\langle v_0, ty_0 \rangle, \langle v_1, ty_1 \rangle, \dots \langle v_n, ty_n \rangle), v) \equiv \neg \exists i: v_i = v$$

The relation *assignable-to*: $Ty \times Ca \times \theta \times D$ indicates whether a particular calculation is assignable to a particular type, based on existing variable types and domain supertype information.

$$\begin{aligned} \text{assignable-to}(ty, calc \in V, \theta, d) \equiv \\ & \neg \text{variable-undefined}(\theta, v) \wedge \text{derived-from}(\text{variable-type}(\theta, calc), ty, d) \\ \text{assignable-to}(ty, o \in O, \theta, d) \equiv & \langle o, ty_s \rangle \in \text{domain-constants}(o) \wedge \text{derived-from}(ty_s, ty, d) \\ \text{assignable-to}(ty, i \in \mathbb{I}, \theta, d) \equiv & \text{derived-from}(ty, \text{INTEGER}, d) \\ \text{assignable-to}(ty, r \in \mathbb{R}, \theta, d) \equiv & \text{derived-from}(ty, \text{REAL}, d) \\ \text{assignable-to}(ty, b \in \{\text{TRUE}, \text{FALSE}\}, \theta, d) \equiv & \text{derived-from}(ty, \text{BOOLEAN}, d) \\ \text{assignable-to}(ty, \langle f, te \rangle \in F \times Te, \theta, d) \equiv & \text{derived-from}(\text{function-value-type}(f), ty, d) \\ \text{assignable-to}(ty, \langle op, calc1, calc2 \rangle \in Op \times Ca \times Ca, \theta, d) \equiv \\ & \text{derived-from}(ty, \text{REAL}, d) \\ & \vee (\text{derived-from}(ty, \text{INTEGER}, d) \\ & \wedge (op = : \text{UNIFORM} \\ & \vee (op \in \{+, -, *, /\} \wedge \text{assignable-to}(\text{INTEGER}, calc1, \theta, d) \wedge \text{assignable-to}(\text{INTEGER}, calc2, \theta, d)))) \end{aligned}$$

$$\begin{aligned}
& assignable\text{-}to(ty, \langle aggop, v, con, calc \rangle \in AggOp \times V \times Cn \times Ca, \theta, d) \equiv \\
& \quad \exists \theta_N \supset \theta: \neg variable\text{-}undefined(\theta_N, v) \\
& \quad \wedge (derived\text{-}from(ty, REAL, d) \wedge assignable\text{-}to(calc, REAL, \theta, d) \\
& \quad \vee derived\text{-}from(ty, INTEGER, d) \wedge assignable\text{-}to(calc, INTEGER, \theta, d))
\end{aligned}$$

$$\begin{aligned}
& assignable\text{-}to(ty, \langle cnd, calc_T, calc_F \rangle \in IF Cn \times Ca \times Ca, \theta, d) \equiv \\
& \quad derived\text{-}from(ty, REAL, d) \wedge assignable\text{-}to(calc_T, REAL, \theta, d) \wedge assignable\text{-}to(calc_F, REAL, \theta, d) \\
& \quad \vee derived\text{-}from(ty, INTEGER, d) \wedge assignable\text{-}to(calc_T, INTEGER, \theta, d) \wedge assignable\text{-}to(calc_F, INTEGER, \theta, d)
\end{aligned}$$

All variables and values are legal terms. A function term is legal if the terms used with it are terms assignable to that term's argument types.

$$\begin{aligned}
& legal\text{-}term(te \in V \cup Va, \theta, \mathbf{v}_{BOUND}, d) \equiv TRUE \\
& legal\text{-}term(te = \langle fn, \mathbf{te} \rangle \in Fn \times \mathbf{Te}, \theta, \mathbf{v}_{BOUND}, d) \equiv \\
& \quad \exists f \in domain\text{-}functions(d): \\
& \quad \quad function\text{-}name(f) = fn \\
& \quad \quad \wedge |function\text{-}arguments(f)| = |\mathbf{te}| \\
& \quad \quad \wedge \forall i, 0 \leq i < |\mathbf{te}|: assignable\text{-}to(function\text{-}argument\text{-}type(f, function\text{-}arguments(f)_i), \mathbf{te}_i, \theta, d) \\
& \quad \quad \wedge legal\text{-}calculation(ty, \mathbf{te}_i, \theta, d)
\end{aligned}$$

A calculation is legal if it's a bound term, bound fluent with correct argument types, or an operation with correct argument types. An aggregation is legal if its constraint condition and subcalculation are legal, and the calculation matches the supertype.

$$\begin{aligned}
& legal\text{-}calculation(ty, te \in Te, \theta, d) \equiv \\
& \quad assignable\text{-}to(ty, te, \theta, d) \\
& \quad \wedge (te \notin V \wedge \neg variable\text{-}undefined(\theta, te)) \\
& legal\text{-}calculation(ty, \langle op, calc1, calc2 \rangle \in Op \times Ca \times Ca, \theta, d) \equiv \\
& \quad assignable\text{-}to(ty, \langle op, calc1, calc2 \rangle, \theta, d) \\
& \quad \wedge ((op \neq :UNIFORM \wedge assignable\text{-}to(REAL, calc1, \theta, d) \wedge assignable\text{-}to(REAL, calc2, \theta, d)) \\
& \quad \vee (op = :UNIFORM \wedge assignable\text{-}to(INTEGER, calc1, \theta, d) \wedge assignable\text{-}to(INTEGER, calc2, \theta, d))) \\
& legal\text{-}calculation(ty, \langle aggop, v, con, calc \rangle \in AggOp \times V \times Cn \times Ca, \theta, d) \equiv \\
& \quad assignable\text{-}to(ty, \langle aggop, v, con, calc \rangle, \theta, d) \\
& \quad \wedge \exists \theta_N: extends(\theta_N, \theta) \\
& \quad \quad variable\text{-}undefined(\theta, v) \wedge \neg variable\text{-}undefined(\theta_N, v) \\
& \quad \quad \wedge legal\text{-}condition(con, \theta_N, d) \\
& \quad \quad \wedge legal\text{-}calculation(ty, calc, \theta_N, d) \\
& legal\text{-}calculation(ty, \langle IF, cnd, calc_T, calc_F \rangle \in IF Cn \times Ca \times Ca, \theta, d) \equiv \\
& \quad assignable\text{-}to(ty, \langle cnd, calc_T, calc_F \rangle, \theta, d) \\
& \quad \wedge legal\text{-}condition(cnd, \theta_N, d) \\
& \quad \wedge legal\text{-}calculation(ty, calc_T, \theta_N, d) \\
& \quad \wedge legal\text{-}calculation(ty, calc_F, \theta_N, d)
\end{aligned}$$

For a condition to be legal, all bindings must have constituent types throughout subconditions. An “and” or “or” condition is legal if its two condition arguments are legal, and a forall condition is legal if its two condition arguments are legal, and its variables are not previously defined. Comparisons with an inequality are legal only if all of the calculation variables are bound. In the case of a single variable being equal to a bound calculation, that variable becomes bound.

$$\text{legal-condition}(te \in Te, \theta, \mathbf{v}_{\text{BOUND}}, d) \equiv$$

$$\text{legal-term}(te, \theta, \mathbf{v}_{\text{BOUND}}, d) \wedge \text{assignable-to}(\text{BOOLEAN}, te, \theta, d)$$

$$\text{legal-condition}(\langle bop, cnd1, cnd2 \rangle \in BOp \times Cn \times Cn, \theta, \mathbf{v}_{\text{BOUND}}, d) \equiv$$

$$\text{legal-condition}(cnd1, \theta, \mathbf{v}_{\text{BOUND}}, d)$$

$$\wedge \text{legal-condition}(cnd2, \theta, \mathbf{v}_{\text{BOUND}}, d)$$

$$\wedge bop \in \{\text{AND}, \text{OR}\}$$

$$\text{legal-condition}(\langle \text{FORALL}, \mathbf{v}, cnd1, cnd2 \rangle, \theta, \mathbf{v}_{\text{BOUND}}, d) \equiv$$

$$\exists \theta_N, \mathbf{v1}_{\text{INNER-BOUND}}, \mathbf{v2}_{\text{INNER-BOUND}}:$$

$$\text{extends}(\theta_N, \theta)$$

$$\wedge \forall v \in \mathbf{v}: \text{variable-undefined}(\theta, v) \wedge \neg \text{variable-undefined}(\theta_N, v)$$

$$\wedge \mathbf{v1}_{\text{INNER-BOUND}} = \text{condition-bound-variables}(cnd1, \mathbf{v1}_{\text{INNER-BOUND}}) \cup \mathbf{v}_{\text{BOUND}} \cup \mathbf{v}$$

$$\wedge \text{legal-condition}(cnd1, \theta_N, \mathbf{v1}_{\text{INNER-BOUND}}, d)$$

$$\wedge \mathbf{v2}_{\text{INNER-BOUND}} = \text{condition-bound-variables}(cnd2, \mathbf{v2}_{\text{INNER-BOUND}}) \cup \mathbf{v1}_{\text{INNER-BOUND}} \cup \mathbf{v}$$

$$\wedge \text{legal-condition}(cnd2, \theta_N, \mathbf{v2}_{\text{INNER-BOUND}}, d)$$

$$\text{legal-condition}(\langle \text{ineq}, \text{calc1}, \text{calc2} \rangle \in \text{Ineq} \times Ca \times Ca, \theta, \mathbf{v}_{\text{BOUND}}, d) \equiv$$

$$\exists ty \in Ty:$$

$$\text{legal-calculation}(ty, \text{calc1}, \text{restrict}(\theta, \mathbf{v}_{\text{BOUND}}), d)$$

$$\wedge \text{legal-calculation}(ty, \text{calc2}, \text{restrict}(\theta, \mathbf{v}_{\text{BOUND}}), d)$$

$$\wedge (\text{derived-from}(\text{REAL}, ty) \vee \text{derived-from}(\text{INTEGER}, ty) \vee \text{ineq} \in \{=, \neq\})$$

$$\wedge (\text{ineq} \in \{=\} \vee (\text{variables-in-calculation}(\text{calc1}) \cup \text{variables-in-calculation}(\text{calc2})) \subset \mathbf{v}_{\text{BOUND}}))$$

$$\text{restrict}(\theta = (\langle v_0, ty_0 \rangle, \dots \langle v_n, ty_n \rangle), \mathbf{v}_{\text{BOUND}}) =$$

$$\text{concat}(\text{if}(v_0 \in \mathbf{v}_{\text{BOUND}}, \text{TRUE}: (\langle v_0, ty_0 \rangle), \text{FALSE}: ()),$$

$$\text{restrict}((\langle v_1, ty_1 \rangle, \dots \langle v_n, ty_n \rangle), \mathbf{v}_{\text{BOUND}}))$$

$$\text{extends}(\theta_B = (\langle v_{B,0}, ty_{B,0} \rangle, \dots \langle v_{B,m}, ty_{B,m} \rangle), \theta_S = (\langle v_{S,0}, ty_{S,0} \rangle, \dots \langle v_{S,m}, ty_{S,m} \rangle)) \equiv$$

$$m > n \wedge \forall i, 0 \leq i \leq n: v_{B,i} = v_{S,i} \wedge ty_{B,i} = ty_{S,i}$$

$$\text{bound-variables-in-conditions}((pc_0 \dots pc_N) \in \mathbf{PC}, \mathbf{v}_{\text{BOUND}}) =$$

$$\cup_i \text{condition-bound-variables}(cnd, pc_i)$$

$$\text{condition-bound-variables}(te \in Te, \mathbf{v}_{\text{BOUND}}) = \emptyset$$

$$\text{condition-bound-variables}(\langle \text{OR}, cnd1, cnd2 \rangle \in \text{OR} \times Cn \times Cn, d, \mathbf{v}_{\text{BOUND}}) = \emptyset$$

$$\text{condition-bound-variables}(\langle \text{AND}, cnd1, cnd2 \rangle \in \text{AND} \times Cn \times Cn, d, \mathbf{v}_{\text{BOUND}}) =$$

$$\text{condition-bound-variables}(cnd1, d, \mathbf{v}_{\text{BOUND}}) \cup \text{condition-bound-variables}(cnd2, d, \mathbf{v}_{\text{BOUND}})$$

$$\text{condition-bound-variables}(\langle \text{FORALL}, \mathbf{v}, cnd1, cnd2 \rangle \in \text{FORALL} \times \mathbf{V} \times Cn \times Cn, d, \mathbf{v}_{\text{BOUND}}) = \emptyset$$

$$\text{condition-bound-variables}(\langle \text{ineq}, \text{calc1}, \text{calc2} \rangle \in \text{Ineq} \times \text{Ca} \times \text{Ca}, d, \mathbf{v}_{\text{BOUND}}) = \begin{cases} \text{calc1} \in V \wedge \text{variables-in-calculation}(\text{calc2}) \subset \mathbf{v}_{\text{BOUND}} & \{ \text{calc1} \} \\ \text{calc2} \in V \wedge \text{variables-in-calculation}(\text{calc1}) \subset \mathbf{v}_{\text{BOUND}} & \{ \text{calc2} \} \\ \text{otherwise} & \emptyset \end{cases}$$

$$\text{variables-in-term}(te \in Va) \equiv \emptyset$$

$$\text{variables-in-term}(te \in V) \equiv \{te\}$$

$$\text{variables-in-term}(te = \langle fn, \mathbf{te} \rangle \in Fn \times \mathbf{Te}) \equiv \bigcup_i \text{variables-in-term}(\mathbf{te}_i)$$

$$\text{variables-in-calculation}(\text{calc} \in Te) \equiv \bigcup_i \text{variables-in-term}(\mathbf{te}_i)$$

$$\begin{aligned} \text{variables-in-calculation}(\text{calc} = \langle op, \text{calc1}, \text{calc2} \rangle \in Op \times Ca \times Ca) &\equiv \\ \text{variables-in-calculation}(\text{calc1}) \cup \text{variables-in-calculation}(\text{calc2}) & \end{aligned}$$

$$\begin{aligned} \text{variables-in-calculation}(\text{calc} = \langle \text{aggop}, v, \text{con}, \text{calcsb} \rangle \in \text{AggOp} \times V \times Cn \times Ca) &\equiv \\ (\text{variables-in-calculation}(\text{calcsb}) \cup \text{variables-in-condition}(\text{con})) / \{v\} & \end{aligned}$$

$$\begin{aligned} \text{variables-in-calculation}(\text{calc} = \langle \text{if}, \text{cnd}, \text{calc1}, \text{calc2} \rangle \in \text{IF} \times Cn \times Ca \times Ca) &\equiv \\ \text{variables-in-condition}(\text{cnd}) \cup \text{variables-in-calculation}(\text{calc1}) \cup \text{variables-in-calculation}(\text{calc2}) & \end{aligned}$$

$$\text{variables-in-condition}(te \in Te) \equiv \text{variables-in-term}(te)$$

$$\begin{aligned} \text{variables-in-condition}(\langle \text{bop}, \text{cnd1}, \text{cnd2} \rangle \in \text{BOp} \times Cn \times Cn) &\equiv \\ \text{variables-in-condition}(\text{cnd1}) \cup \text{variables-in-condition}(\text{cnd2}) & \end{aligned}$$

$$\begin{aligned} \text{variables-in-condition}(\langle \text{forall}, \mathbf{v}, \text{cnd1}, \text{cnd2} \rangle \in \text{FORALL} \times \mathbf{V} \times Cn \times Cn) &\equiv \\ \text{variables-in-condition}(\text{cnd1}) \cup \text{variables-in-condition}(\text{cnd2}) / \{v\} & \end{aligned}$$

$$\begin{aligned} \text{variables-in-condition}(\langle \text{ineq}, \text{calc1}, \text{calc2} \rangle \in \text{Ineq} \times Ca \times Ca) &\equiv \\ \text{variables-in-calculation}(\text{calc1}) \cup \text{variables-in-calculation}(\text{calc2}) & \end{aligned}$$

$$\begin{aligned} \text{effect-arguments-bound}(e, d, \mathbf{v}_{\text{BOUND}}) &= \\ (\text{effect-argument-list}(e, d) \cap V) \subset \mathbf{v}_{\text{BOUND}} & \end{aligned}$$

$$\begin{aligned} \text{effect-argument-list}(e, d) &= \\ \{te \mid \exists f, v: f \in \text{domain-functions}(d) \wedge \text{function-name}(f) = \text{effect-name}(e) \\ &\quad \wedge v \in \text{function-arguments}(f) \wedge \text{effect-argument}(e, v) = te\} \end{aligned}$$

$$\begin{aligned} \text{change-arguments-bound}(c, d, \mathbf{v}_{\text{BOUND}}) &= \\ (\text{change-argument-list}(c, d) \cap V) \subset \mathbf{v}_{\text{BOUND}} & \end{aligned}$$

$$\begin{aligned} \text{change-argument-list}(c, d) &= \\ \{te \mid \exists f, v: f \in \text{domain-functions}(d) \wedge \text{function-name}(f) = \text{change-name}(c) \\ &\quad \wedge v \in \text{function-arguments}(f) \wedge \text{change-argument}(c, v) = te\} \end{aligned}$$

$$\begin{aligned} \text{axiom-argument-list}(ax, d) &= \\ \{te \mid \exists f, v: f \in \text{domain-functions}(d) \wedge \text{function-name}(f) = \text{axiom-name}(ax) \\ &\quad \wedge v \in \text{function-arguments}(f) \wedge \text{axiom-argument}(ax, v) = te\} \end{aligned}$$

Effects are legal if and only if they match a fluent properly, and match necessary variable type bindings θ .

$$\begin{aligned}
& \text{legal-effect}(e, \theta, \mathbf{v}_{\text{BOUND}}, d) \equiv \\
& \exists f \in \text{domain-functions}(d): \\
& \quad \text{function-name}(f) = \text{effect-name}(e) \\
& \quad \wedge (\text{derived-from}(\text{REAL}, \text{function-value-type}(f)) \vee \text{derived-from}(\text{INTEGER}, \text{function-value-type}(f)) \\
& \quad \vee \text{effect-modification}(e) = \text{SET}) \\
& \quad \wedge \text{assignable-to}(\text{function-value-type}(f), \text{effect-value}(e), \theta, d) \\
& \quad \wedge (\{\text{effect-value}(e)\} \cap V) \subseteq \mathbf{v}_{\text{BOUND}} \\
& \quad \wedge \text{effect-arguments-bound}(e, \mathbf{v}_{\text{BOUND}}, d) \\
& \quad \wedge \forall v \in \text{function-arguments}(f): \text{assignable-to}(\text{effect-argument}(e, v), \text{function-argument-type}(f, v), \theta, \\
& \quad d)
\end{aligned}$$

Continuous changes are legal if and only if they match a function properly, as well as variable types θ .

$$\begin{aligned}
& \text{legal-change}(c, \theta, d) \equiv \\
& \exists f \in \text{domain-functions}(d): \\
& \quad \text{function-name}(f) = \text{change-name}(c) \\
& \quad \wedge (\text{derived-from}(\text{REAL}, \text{function-value-type}(f)) \vee \text{derived-from}(\text{INTEGER}, \text{function-value-type}(f))) \\
& \quad \wedge \text{legal-calculation}(\text{function-value-type}(f), \text{change-value}(c), \theta, d) \\
& \quad \wedge \text{change-arguments-bound}(c, \mathbf{v}_{\text{BOUND}}, d) \\
& \quad \wedge \forall v \in \text{function-arguments}(f): \text{assignable-to}(\text{change-argument}(c, v), \text{function-argument-type}(f, v), \\
& \quad \theta, d)
\end{aligned}$$

$$\begin{aligned}
& \text{legal-function}(f, d) \equiv \\
& \quad \forall v \in \text{function-arguments}(f): \text{function-argument-type}(f, v) \in \text{domain-types}(d) \\
& \quad \wedge \text{function-value-type}(f) \in \text{domain-types}(d)
\end{aligned}$$

$$\begin{aligned}
& \text{legal-axiom}(ax, \theta, d) \equiv \\
& \exists \theta, f, \mathbf{v}_{\text{BOUND}}: \\
& \quad f \in \text{domain-functions}(d) \\
& \quad \wedge \text{function-name}(f) = \text{axiom-name}(ax) \\
& \quad \wedge \mathbf{v}_{\text{BOUND}} = (\text{axiom-argument-list}(ax) \cap V) \\
& \quad \wedge \text{derived-from}(\text{BOOLEAN}, \text{function-value-type}(f)) \\
& \quad \wedge \text{legal-condition}(\text{axiom-antecedent}(f), \theta, \mathbf{v}_{\text{BOUND}}, d) \\
& \quad \wedge \forall v \in \text{function-arguments}(f): \\
& \quad \quad \text{assignable-to}(\text{function-argument-type}(f, v), \text{axiom-argument}(ax, v), \theta, d)
\end{aligned}$$

$$\begin{aligned}
& \text{legal-action}(a, d) \equiv \\
& \exists \theta, \mathbf{v}_{\text{BOUND}}: \\
& \quad \forall v \in \text{action-parameters}(a): (\text{assignable-to}(\text{action-parameter-type}(a, v), v, \theta, d) \\
& \quad \quad \wedge \text{action-parameter-type}(a, v) \in \text{domain-types}(d)) \\
& \quad \wedge \text{assignable-to}(\text{AGENT}, \text{action-performer}(a), \theta, d) \\
& \quad \wedge \mathbf{v}_{\text{BOUND}} = \text{bound-variables-in-conditions}(\text{cnd}, \mathbf{v}_{\text{BOUND}}) \\
& \quad \quad \cup \text{action-parameters}(a) \cup (\{\text{action-performer}(a)\} \cap V) \\
& \quad \wedge \forall \text{cnd} \in \text{action-preconditions}(a): \text{legal-condition}(\text{cnd}, \theta, \mathbf{v}_{\text{BOUND}}, d) \\
& \quad \wedge \forall e \in \text{action-effects}(a): \text{legal-effect}(e, \theta, \mathbf{v}_{\text{BOUND}}, d)
\end{aligned}$$

$legal-event(ev, d) \equiv$
 $\exists \theta, \mathbf{v}_{BOUND}:$
 $\forall v \in event-qualities(ev): (assignable-to(event-quality-type(ev, v), v, \theta, d)$
 $\quad \wedge event-quality-type(ev, v) \in domain-types(d))$
 $\wedge \mathbf{v}_{BOUND} = bound-variables-in-conditions(cnd, \mathbf{v}_{BOUND}) \cup event-qualities(ev)$
 $\wedge \forall cnd \in event-triggers(ev): legal-condition(cnd, \theta, \mathbf{v}_{BOUND}, d)$
 $\wedge \forall e \in event-effects(ev): legal-effect(e, \theta, \mathbf{v}_{BOUND}, d)$
 $legal-process(p, d) \equiv$
 $\exists \theta, \mathbf{v}_{BOUND}: \forall v \in process-qualities(p):$
 $(assignable-to(process-quality-type(p, v), v, \theta, d)$
 $\quad \wedge process-quality-type(p, v) \in domain-types(d))$
 $\wedge \mathbf{v}_{BOUND} = bound-variables-in-conditions(cnd, \mathbf{v}_{BOUND}) \cup process-qualities(p)$
 $\wedge \forall cnd \in process-conditions(p): legal-condition(cnd, \theta, \mathbf{v}_{BOUND}, d)$
 $\wedge \forall c \in process-changes(p): legal-change(c, \theta, \mathbf{v}_{BOUND}, d)$
 $legal-domain(d) \equiv$
 $\forall \langle child, parent \rangle \in domain-supertypes(d): child \in domain-types(d) \wedge parent \in domain-types(d)$
 $\forall \langle object, type \rangle \in domain-constants(d): type \in domain-types(d)$
 $\wedge \forall f \in domain-functions(d): legal-function(f, d)$
 $\wedge \forall a \in domain-actions(d): legal-action(a, d)$
 $\wedge \forall ev \in domain-events(d): legal-event(ev, d)$
 $\wedge \forall p \in domain-processes(d): legal-process(p, d)$
 $\wedge \forall ax \in domain-axioms(d): legal-axiom(ax, d)$
 $legal-ground-fluent(gf = \langle fn, \mathbf{va}_{ARGS}, \mathbf{va}_{FV} \rangle \in Fn \times \mathbf{Va} \times \mathbf{Va}, d) \equiv$
 $\exists f \in domain-functions(d):$
 $function-name(f) = fn$
 $\wedge |function-arguments(f)| = |\mathbf{va}_{ARGS}|$
 $\wedge \forall i, 0 \leq i < |\mathbf{va}_{ARGS}|: assignable-to(function-argument-type(f, function-arguments(f)_i), \mathbf{va}_{ARGS}_i, \emptyset, d)$
 $\quad \wedge assignable-to(function-value-type(f), \mathbf{va}_{FV}, \emptyset, d) legal-environment(d, sg) \equiv$
 $legal-domain(d)$
 $\wedge \forall gf \in scenario-generator-fluents(sg):$
 $\wedge \forall \langle fn, val \rangle \in scenario-generator-defaults(sg): \exists f \in domain-functions(d): function-name(f) = fn$
 $\quad \wedge assignable-to(val, function-value-type(f), \emptyset, d)$
 $\wedge \forall f \in domain-functions(d): \exists val: \langle function-name(f), val \rangle \in scenario-generator-defaults(sg)$
 $\wedge \forall og \in scenario-generator-object-generators(g): derived-from(object-generator-type(og), OBJECT, d)$
 $\wedge \forall fg \in scenario-generator-fluent-generators(sg):$
 $\exists f: f \in domain-functions(d) \wedge function-name(f) = fluent-generator-function-name(fg)$
 $\quad \wedge G(legal-ground-fluent(fluent-generator-draw(fg), \emptyset, \emptyset, d))$
 $\wedge legal-calculation(\mathbf{REAL}, scenario-generator-performance-calculation(sg), \{\{AG, AGENT\}\}, d)$

Appendix B Definition of *apply* function

We can now define the *apply* function for each of a set of transformations.

apply(ADDTYPE (TYPE: *new-type*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types} \cup \{\text{new-type}\}, \text{supertypes},$
 $\text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

apply(ADDTYPEPARENT(CHILD: *child*, PARENT: *parent*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types}, \text{supertypes} \cup \{\langle \text{child}, \text{parent} \rangle\},$
 $\text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

apply(REMOVETYPEPARENT(CHILD: *child*, PARENT: *parent*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types}, \text{supertypes} \setminus \{\langle \text{child}, \text{parent} \rangle\},$
 $\text{constantTypes}, \text{functions}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

apply(ADDCONSTANT(NAME: *name*, TYPE: *type*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types}, \text{supertypes}, \text{constantTypes} \cup \{\langle \text{name}, \text{type} \rangle\},$
 $\text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

apply(ADDFUNCTION(FUNCTION: *new-function*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions} \cup \{\text{new-function}\},$
 $\text{axioms}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

apply(ADDAXIOM(AXIOM: *axiom*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions},$
 $\text{axioms} \cup \{\text{axiom}\}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

apply(REMOVETYPE(NAME: *name*),
 $\langle \text{types}, \text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes} \rangle, \text{sg}$) $\equiv$
 $\langle \text{types} \setminus \text{name},$
 $\text{supertypes}, \text{constantTypes}, \text{functions}, \text{axioms}, \text{actions}, \text{events}, \text{processes}, \text{perf} \rangle$

$apply(REMOVEFUNCTION(NAME: name),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes,$
 $\{f \in functions \mid function-name(f) \neq name\}, axioms, actions, events, processes, perf \rangle$

$apply(REMOVEAXIOM(AXIOM: axiom),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions,$
 $axioms \setminus \{axiom\}, actions, events, processes, perf \rangle$

$apply(ADDACTION(ACTIONNAME: name, PERFORMER: performer, PARAMETERS: params,$
 $PARAMETERTYPES: paramTypes),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $actions \cup \{\langle name, performer, args, argTypes, [], [] \rangle\}, events, processes, perf \rangle$

$apply(REMOVEACTION(ACTIONNAME: name),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $\{a \in actions \mid action-name(a) \neq name\}, events, processes, perf \rangle$

$apply(t = ADDPRECONDITION(ACTIONNAME: name, PRECONDITION: new-prec),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $\{applyToAction(t, a) \mid a \in actions\}, events, processes, perf \rangle$

$applyToAction(ADDPRECONDITION(ACTIONNAME: name, PRECONDITION: new-prec),$
 $\langle name, perf, args, argTypes, precs, effs \rangle) \equiv$
 $\langle name, perf, args, argTypes, precs \cup \{new-prec\}, effs \rangle$

$applyToAction(ADDPRECONDITION(ACTIONNAME: name, PRECONDITION: new-prec),$
 $a = \langle n \neq name, perf, args, argTypes, precs, effs \rangle) \equiv a$

$apply(t = REMOVEPRECONDITION(ACTIONNAME: name, PRECONDITION: old-prec),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, \{applyToAction(t, a) \mid a \in actions\},$
 $events, processes, perf \rangle$

$applyToAction(REMOVEPRECONDITION(ACTIONNAME: name, PRECONDITION: old-prec),$
 $\langle name, args, perf, precs, effs \rangle) \equiv$
 $\langle name, perf, args, argTypes, precs \setminus \{old-prec\}, effs \rangle$

applyToAction(REMOVEPRECONDITION(ACTIONNAME: *name*, PRECONDITION: *old-prec*),
 $a = \langle n \neq name, perf, args, argTypes, precs, effs \rangle \equiv a$)

apply($t = \text{ADDACTIONEFFECT}(\text{ACTIONNAME: } name, \text{EFFECT: } new\text{-effect})$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $\{applyToAction(t, a) \mid a \in actions\}, events, processes, perf \rangle$

applyToAction(ADDACTIONEFFECT(ACTIONNAME: *name*, EFFECT: *new-effect*),
 $\langle name, perf, args, argTypes, precs, effs \rangle \equiv$
 $\langle name, perf, args, argTypes, precs, effs \cup \{new\text{-effect}\} \rangle$)

applyToAction(ADDACTIONEFFECT(ACTIONNAME: *name*, EFFECT: *new-effect*),
 $a = \langle n \neq name, perf, args, argTypes, precs, effs \rangle \equiv a$)

apply($t = \text{REMOVEACTIONEFFECT}(\text{ACTIONNAME: } name, \text{EFFECT: } old\text{-effect})$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $\{applyToAction(t, a) \mid a \in actions\}, ev \in events, processes, perf \rangle$

applyToAction(REMOVEACTIONEFFECT(ACTIONNAME: *name*, EFFECT: *old-effect*),
 $\langle name, perf, args, argTypes, precs, effs \rangle \equiv$
 $\langle name, perf, args, argTypes, precs, effs \setminus \{old\text{-effect}\} \rangle$)

applyToAction(REMOVEACTIONEFFECT(ACTIONNAME: *name*, EFFECT: *old-effect*),
 $a = \langle n \neq name, perf, args, argTypes, precs, effs \rangle \equiv a$)

apply($t = \text{ADDEVENT}(\text{EVENTNAME: } name, \text{QUALITIES: } quals, \text{QUALITYTYPES: } qualTypes)$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions,$
 $events \cup \{\langle name, quals, qualTypes, 1, 0, \emptyset, \emptyset \rangle\}, processes, perf \rangle$

apply(REMOVEEVENT(NAME: *name*),
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions,$
 $\{ev \in events \mid event\text{-name}(ev) \neq name\}, processes, perf \rangle$)

apply($t = \text{CHANGEFREQUENCY}(\text{EVENTNAME: } name, \text{FREQUENCY: } new\text{-freq})$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions,$
 $\{applyToEvent(t, ev) \mid ev \in events\}, processes \cup \{new\text{-process}\} \rangle$

applyToEvent(CHANGEFREQUENCY(EVENTNAME: *name*, FREQUENCY: *new-freq*),
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv$
 $\langle name, quals, qualTypes, prob, new-freq, triggers, effs \rangle$)

applyToEvent(CHANGEFREQUENCY(EVENTNAME: *name*, FREQUENCY: *new-freq*),
 $ev = \langle n \neq name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv ev$)

apply($t = \text{CHANGEPROBABILITY}(\text{EVENTNAME: } name, \text{PROBABILITY: } new-prob)$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions,$
 $\{applyToEvent(t, ev) \mid ev \in events\}, processes \rangle$)

applyToEvent(CHANGEPROBABILITY(EVENTNAME: *name*, PROBABILITY: *new-prob*),
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv$
 $\langle name, quals, qualTypes, new-prob, freq, triggers, effs \rangle$)

applyToEvent(CHANGEPROBABILITY(EVENTNAME: *name*, PROBABILITY: *new-prob*),
 $ev = \langle n \neq name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv ev$)

apply($t = \text{ADDTIGGER}(\text{EVENTNAME: } name, \text{PRECONDITION: } new-prec)$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $actions, \{applyToEvent(t, ev) \mid ev \in events\}, processes, perf \rangle$)

applyToEvent(ADDTIGGER(EVENTNAME: *name*, PRECONDITION: *new-prec*),
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv$
 $\langle name, quals, qualTypes, prob, prec \cup \{new-prec\}, effs \rangle$)

applyToEvent(ADDTIGGER(EVENTNAME: *name*, PRECONDITION: *new-prec*),
 $ev = \langle n \neq name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv ev$)

apply($t = \text{REMOVETRIGGER}(\text{EVENTNAME: } name, \text{TRIGGER: } old-trigger)$,
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions,$
 $\{applyToEvent(t, ev) \mid ev \in events\}, processes, perf \rangle$)

applyToEvent(REMOVETRIGGER(EVENTNAME: *name*, TRIGGER: *old-trigger*),
 $\langle name, quals, qualTypes, prob, triggers, effs \rangle \equiv$
 $\langle name, quals, qualTypes, prob, freq, triggers \setminus \{old-trigger\}, effs \rangle$)

applyToEvent(REMOVETRIGGER(EVENTNAME: *name*, TRIGGER: *old-trigger*),
 $ev = \langle n \neq name, quals, qualTypes, prob, freq, triggers, effs \rangle \equiv ev$)

$apply(t = ADDEVENTEFFECT(EVENTNAME: name, EFFECT: new-effect),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $actions, \{applyToEvent(t, ev) \mid ev \in events\}, processes, perf \rangle$

$applyToEvent(ADDEVENTEFFECT(EVENTNAME: name, EFFECT: new-effect),$
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \rangle) \equiv$
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \cup \{new-effect\} \rangle$

$applyToEvent(ADDEVENTEFFECT(EVENTNAME: name, EFFECT: new-effect),$
 $ev = \langle n \neq name, quals, qualTypes, prob, freq, triggers, effs \rangle) \equiv ev$

$apply(t = REMOVEEVENTEFFECT(EVENTNAME: name, EFFECT: old-effect),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms,$
 $actions, \{applyToEvent(t, ev) \mid ev \in events\}, processes, perf \rangle$

$applyToEvent(REMOVEEVENTEFFECT(EVENTNAME: name, EFFECT: old-effect),$
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \rangle) \equiv$
 $\langle name, quals, qualTypes, prob, freq, triggers, effs \setminus \{old-effect\} \rangle$

$applyToEvent(REMOVEEVENTEFFECT(EVENTNAME: name, EFFECT: old-effect),$
 $ev = \langle n \neq name, quals, qualTypes, prob, freq, triggers, effs \rangle) \equiv ev$

$apply(ADDPROCESS(PROCESSNAME: name, QUALITIES: quals, QUALITYTYPES: qualTypes),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events,$
 $processes \cup \{\langle name, quals, qualTypes, \emptyset, \emptyset \rangle\}, perf \rangle$

$apply(REMOVEPROCESS(NAME: name),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events,$
 $\{p \in processes \mid process-name(p) \neq name\} \rangle$

$apply(t = ADDPROCESSCONDITION(PROCESSNAME: name, CONDITION: new-condition),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events,$
 $\{applyToProcess(t, p) \mid p \in processes\} \rangle$

$applyToProcess(ADDPROCESSCONDITION(PROCESSNAME: name, CONDITION: new-condition),$
 $\langle name, quals, qualTypes, conds, changes \rangle) \equiv$
 $\langle name, quals, qualTypes, conds \cup \{new-condition\}, changes \rangle$

$applyToProcess(ADDPROCESSCONDITION(PROCESSNAME: name, CONDITION: new-condition),$
 $p = \langle n \neq name, quals, qualTypes, conds, changes \rangle) \equiv p$

$apply(t = REMOVEPROCESSCONDITION(PROCESSNAME: name, CONDITION: old-condition),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events,$
 $\{applyToProcess(t, p) \mid p \in processes\} \rangle$

$applyToProcess(REMOVEPROCESSCONDITION(PROCESSNAME: name, CONDITION: old-condition),$
 $\langle name, quals, qualTypes, conds, changes \rangle) \equiv$
 $\langle name, quals, qualTypes, conds \setminus \{old-condition\}, changes \rangle$

$applyToProcess(REMOVEPROCESSCONDITION(PROCESSNAME: name, CONDITION: old-condition),$
 $p = \langle n \neq name, quals, qualTypes, conds, changes \rangle) \equiv p$

$apply(t = ADDPROCESSCHANGE(PROCESSNAME: name, CHANGE: new-change),$
 $\langle types, supertypes, constantTypes, functions, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events,$
 $\{applyToProcess(t, p) \mid p \in processes\} \rangle$

$applyToProcess(ADDPROCESSCHANGE(PROCESSNAME: name, CHANGE: new-change),$
 $\langle name, quals, qualTypes, conds, changes \rangle) \equiv$
 $\langle name, quals, qualTypes, conds, changes \setminus \{new-change\} \rangle$

$applyToProcess(ADDPROCESSCHANGE(PROCESSNAME: name, CHANGE: new-change),$
 $p = \langle n \neq name, quals, qualTypes, conds, changes \rangle) \equiv p$

$apply(t = REMOVEPROCESSCHANGE(PROCESSNAME: name, CHANGE: old-change),$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events, processes \rangle, sg) \equiv$
 $\langle types, supertypes, constantTypes, functions, axioms, actions, events,$
 $\{applyToProcess(t, p) \mid p \in processes\} \rangle$

$applyToProcess(REMOVEPROCESSCHANGE(PROCESSNAME: name, CHANGE: old-change),$
 $\langle name, quals, qualTypes, conds, changes \rangle) \equiv$
 $\langle name, quals, qualTypes, conds, changes \setminus \{old-change\} \rangle$

$applyToProcess(REMOVEPROCESSCHANGE(PROCESSNAME: name, CHANGE: old-change),$
 $p = \langle n \neq name, quals, qualTypes, conds, changes \rangle) \equiv p$

Appendix C Differences between T-SAL Domain Language & PDDL+

T-SAL-CR is closely related to PDDL+, due to its use of events and processes to model mixed discrete-continuous domains in a general way. While there is not yet a complete semantics for simulating T-SAL, we expect that one will be provided later. The following differences between T-SAL and PDDL+ are relevant to understanding the language:

- **CREATE effects:** Actions and events can result in the creation of new objects. Open worlds need to allow for new objects to arise as part of the dynamics.
- **Object fluents:** T-SAL makes use of object fluents, not just numeric fluents. Object fluents make many expressions more succinct and permit the description of one-to-one relationships between objects.
- **Calculations:** These are comparable to PDDL+ fluent expressions; T-SAL adds mathematical aggregation over a class of objects, a trinary “if” operator, and recursion through function terms.
- **Conditionals:** These are comparable to PDDL+ goal descriptions; T-SAL adds recursion through function terms.
- **Process changes:** T-SAL allows arbitrary time differential equations in process changes, whereas PDDL+ incorporates only linear differentials. T-SAL uses DT instead of $\#T$ to signify the differential operator, which is cosmetic.
- **Probabilistic events:** Whereas PDDL+ events occur deterministically, T-SAL events occur nondeterministically, either according to a probability or at times drawn from a distribution.
- **Action performers:** To describe multi-agent domains, T-SAL adds a performer property to agent models that allows preconditions that refer to the agent performing an action.
- **Name changes:** To reflect the fact that event and process variables are outside of agent control, we rename “parameters” to “qualities”. To reflect the fact that the occurrence of an event is required, we use the term “triggers” instead of “preconditions”. Processes have “conditions” instead of “preconditions” or “triggers” as they must be active for an indefinite period.
- **Conditional and probabilistic effects:** Not present in T-SAL, to avoid redundancies with events.
- **“Exists” goals:** Not present in T-SAL, as all free variables are implicitly existentially quantified.
- **“Forall” effects:** Not present in T-SAL, events are intended to handle changes to all objects in a class.
- **Durative actions:** Not present in T-SAL, these are redundant with processes and events.
- **Predicates:** Not present in T-SAL, redundant with fluents.
- **“Either” types:** Not present in T-SAL.

Appendix D T-SAL Computational Representation Backus-Naur Form

<DOMAIN>	::= (DEFINE (DOMAIN <NAME>) [<TYPES-DEF>] [<SUPERTYPES-DEF>] [<CONSTANTS-DEF>] [<FUNCTIONS-DEF>] [<STRUCTURE-DEF>*])
<TYPES-DEF>	::= (:TYPES <TYPED LIST (TYPE)>*)
<CONSTANTS-DEF>	::= (:CONSTANTS <TYPED LIST (OBJECT)>)
<FUNCTIONS-DEF>	::= (:FUNCTIONS <TYPED LIST (FUNCTION-DEF)>)
<FUNCTION-DEF>	::= (<FUNCTION-NAME> <TYPED LIST (VARIABLE)>*)
<STRUCTURE-DEF>	::= <ACTION-MODEL-DEF>
<STRUCTURE-DEF>	::= <EVENT-MODEL-DEF>
<STRUCTURE-DEF>	::= <PROCESS-MODEL-DEF>
<STRUCTURE-DEF>	::= <AXIOM-DEF>
<ACTION-MODEL-DEF>	::= (:ACTION <NAME> :PERFORMER <VARIABLE> :PARAMETERS (<TYPED-LIST (VARIABLE)>) :PRECONDITIONS (<CONDITION>*) :EFFECTS (<EFFECT-DEF>*))
<EVENT-MODEL-DEF>	::= (:EVENT <NAME> [:PROBABILITY <REAL>] [:FREQUENCY <REAL>] :QUALITIES (<TYPED-LIST (VARIABLE)>) :TRIGGERS (<CONDITION>*) :EFFECTS (<EFFECT-DEF>*))
<PROCESS-MODEL-DEF>	::= (:PROCESS <NAME> :QUALITIES (<TYPED-LIST (VARIABLE)>) :CONDITIONS (<CONDITION>*) :CHANGES (<CHANGE-DEF>*))
<AXIOM-DEF>	::= (:- <FUNCTION-CALL> <CONDITION>)
<CONDITION>	::= <TERM>
<CONDITION>	::= (<INEQUALITY> <TERM> <TERM>)
<CONDITION>	::= (OR AND <CONDITION> <CONDITION>+)
<CONDITION>	::= (NOT <CONDITION>)
<CONDITION>	::= (FORALL (VARIABLE+) <CONDITION> <CONDITION>)
<CALCULATION>	::= <TERM>
<CALCULATION>	::= <FLUENT-REF>
<CALCULATION>	::= (<MOP> <CALCULATION> <CALCULATION>)
<CALCULATION>	::= (SUM PRODUCT <VARIABLE> <CONDITION> <CALCULATION>)
<CALCULATION>	::= (IF <CONDITION> <CALCULATION> <CALCULATION>)
<EFFECT-DEF>	::= (<MOD> (<FUNCTION-NAME> <BASIC-TERM>*) <TERM>) [<PROBABILITY>]
<CHANGE-DEF>	::= (<DIR> (<FUNCTION-NAME> <BASIC-TERM>*) <CALCULATION>)
<TERM>	::= <BASIC-TERM>
<TERM>	::= <VARIABLE>
<TERM>	::= (<FUNCTION-NAME> <TERM>*)

<BASIC-TERM>	::= <OBJECT>
<BASIC-TERM>	::= <NUMBER>
<BASIC-TERM>	::= TRUE FALSE
<MOP>	::= + - * / :UNIFORM :GAUSSIAN
<MOD>	::= SET INCREASE DECREASE CREATE
<DIR>	::= INCREASE DECREASE
<TYPED LIST (X)>	::= X*
<TYPED LIST (X)>	::= X ⁺ - <TYPE> <TYPED LIST (X)>*
<TYPE>	::= <NAME>
<FUNCTION-NAME>	::= <NAME>
<OBJECT>	::= <NAME>
<VARIABLE>	::= ?<NAME>
<INEQUALITY>	::= "=" "!=" "<" ">" "<=" ">="

Table 4. Backus-Naur Form specification of T-SAL-CR

Human in the Loop Novelty Generation



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

Human in the Loop Novelty Generation

Mark Bercasio¹, Allison Wong², Dustin Dannenhauer³

¹PacMar Technologies, mbercasio@pacmartech.com

²PacMar Technologies, awong@pacmartech.com

³Parallax Advanced Research, dustin.dannenhauer@parallaxresearch.com

Abstract

Developing artificial intelligence approaches to overcome novel, unexpected circumstances is a difficult, unsolved problem. One challenge to advancing the state of the art in novelty accommodation is the availability of testing frameworks for evaluating performance against novel situations. Recent novelty generation approaches in domains such as Science Birds and Monopoly leverage human domain expertise during the search to discover new novelties. Such approaches introduce human guidance before novelty generation occurs and yield novelties that can be directly loaded into a simulated environment. We introduce a new approach to novelty generation that uses abstract models of environments (including simulation domains) that do not require domain-dependent human guidance to generate novelties. A key result is a larger, often infinite space of novelties capable of being generated, with the trade-off being a requirement to involve human guidance to select and filter novelties post generation. We describe our Human-in-the-Loop novelty generation process using our open-source novelty generation library to test baseline agents in two domains: Monopoly and VizDoom. Our results shows the Human-in-the-Loop method enables users to develop, implement, test, and revise novelties within 4 hours for both Monopoly and VizDoom domains.

1 Introduction

There has been increased interest in artificial intelligence (AI) approaches capable of detecting, characterizing, and accommodating novel situations; situations that violate implicit or explicit assumptions about agents (Senator 2019). AI systems trained to perform tasks in a particular environment rely on large amounts of training data or numerous interactions with an environment simulator. When the environment changes in unexpected ways, these AI approaches typically fails and this is especially true when transitioning an AI system into real-world settings.

Developing AI systems capable of handling novel situations requires new methods and procedures than those that currently exists. Current effort is spent on getting AI solutions that can solve the original task at hand, achieving robust performance without novelty present. However, it is speculated that a range of additional capabilities are needed in AI systems to accommodate novelty, beyond what is needed to learn to solve a task by itself.

In most software development practices, tests are created by human users, therefore an automated test-driven development is gaining interest. Also, the verification and validation community has developed processes to obtain requirements of an AI system from the stakeholders of that systems' development in order to understand exactly what the system needs to be able to achieve to be useful. In all these cases, the tests that are used to evaluate an AI's ability come from humans and the task specification encoded in software. These test cases are known ahead of time. We have clear approaches to testing AI systems in this manner.

Current approaches uses procedural generation to create novelty with the help of a human expert experienced in the domain. This approach have a key advantage in generating more scenarios compared to hand written test cases by human. They remain limited because they are guided primarily from human expert knowledge about the domain, and thus the space of novelties they can generate are limited and biased within the scope of the human. Humans generally have good intuition about bad novelties, defined as irrelevant, noticeable, or controllable (Dannenhauer et al. 2022). By having a human expert build targeted, automatic scenario generators for domains in which they have expert knowledge helps avoid generating bad or irrelevant novelties.

We developed a new human in the loop novelty generation process that leverages human domain experts intuition about what makes a good novelty with a new automatic novelty generation process that improves on the efficiency of previous solutions. We briefly introduce our novelty generator in the next section. Interested readers may refer to (Molineaux, Dannenhauer, and Kildebeck 2023) for more details on that system. Additionally our novelty generator is available on Github¹.

The primary contributions of this paper includes:

1. A 6-step process by which a human domain expert with developer skills can leverage our novelty generator tool to save time in generating novelties, and consider a greater range of novelties with a reduced bias from human imagination.
2. An experimental setup showing that novelties generated in this way are capable of challenging agents across multiple environments.

¹<https://github.com/Parallax-Advanced-Research/noveltygen>

3. Experimental results from two domains: Monopoly and VizDoom
4. A qualitative review of the experience for human developers to use our novelty generator.

Section 2 describes our approach in the context of other similar novelty generation approaches. In Section 3 we briefly cover our novelty generation tool, including how to run it, what the output looks like, etc. In section 4 we describe the human-in-the-loop process where a human uses the novelty generator. In Section 5 we describe our experimental setup where we used the novelty generator to generate novelties in two domains: Monopoly and Vizdoom. Section 6 presents results from the empirical evaluations of the novelties while Section 7 describes a qualitative review of the pros and cons of the human in the loop process from our developers who followed the process to generate the novelties described in Sections 5 and 6. Finally we conclude the paper with a discussion for future work.

2 Related Work

Origins of automatic novel environment generators can be traced back to Metagame (PELL 1992) and EGGG (orw 2000). Both were designed to find novel variations of chess-like games. More recent work in game design research decoupled the design of a game space from the exploration of that space to find interesting games (Smith and Mateas 2010). In their work, the language used to design game spaces was large and flexible compared to the prior chess-like games, but still incorporated many assumptions, such as the existence of a rectangular play space. Even newer work on algorithmic procedural content generation for games focuses on bounded spaces such as levels, characters and quests (Risi and Togelius 2020). In these systems some domain knowledge is encoded by humans to guide the search towards producing valuable artifacts.

Current benchmark domains to evaluate AI agent’s novelty-handling capability are designed in close alignment with simulated environments. NovelGridworlds (Goel et al. 2021) is an OpenAI Gym environment that leverages human guidance on grid-based environments. Another grid-based novelty generator uses an ontology of novelties related to sequential decision making, such as distinguishing between object and action novelties (Balloch et al. 2022). Birds (an Angry Birds AI domain; (Xue et al. 2022)) implements novelties as modifications to the C# codebase. This includes changing physical parameters and colours of existing game objects to more difficult modifications such as introducing a new class of objects (e.g. hostile external agents that hinders the agent). GNOME (Generating Novelty in Open-World Multi-agent Environments; (Kejriwal and Thomas 2021)) is a simulation platform developed to test AI agent’s response to novelty in the classic Monopoly board game, using a library of novelty- generating methods. An example of this is replacing existing game event functions (like a player passing go and collecting \$200) with human inspired function. Functions maintain a similar signature enabling them to be easily swapped out with the original function (such as a function that takes a wealth tax from the player).

3 Novelty Generation Tool

Our approach to novelty generation is originally inspired by Wiggins’ 2006 paper on the Creative Systems Framework. Novelties are created using two types of transformations: R-transformations that change the space of possible states and state transitions, and T-transformations that modify the search method for generating starting states in that environment.

R-transformations² modify a domain file describing the state-transition system of an environment. This includes possible states with symbolic constructs like object type hierarchies, relations between object types, object properties, and fluents. Changes to these structures in effect changes the state space. Domains also define the set of state-transitions that are valid in an environment using action, event, and process models. Changes to these models changes the transition space between states. All changes of this nature (either to the state space or transition space) fall under R-transformations.

Our environment-transformation based novelty generator that we present here is based on domain-independent environments models. Using a formal language inspired from the AI Planning community, Planning Definition Domain language (PDDL), we can generate novelties for any domain capable of being modelled with the language. For the full list of constructs that our Transformation Simulation Abstraction Language (TSAL) supports, refer to (Dannenhauer et al. 2022). A key benefit of our novelty generator is that it is domain-independent since it can generate novelties for any environment modeled in the TSAL language; however the domain-independent novelties it produces needs to be guided. We have proposed domain independent heuristics for filtering novelties on three conditions:

Relevant: The novelty affects the agents performance on the task.

Noticeable: The novelty causes different observations to the agent.

Controllable: The novelty rewards agent policies differently.

In (Dannenhauer et al. 2022) we show that using a simple re-planning agent and a simulated version of a TSAL environment, novelties can be automatically filtered out based on relevance, with the rest left for future work.

Our novelty generator is implemented in python, available on GitHub, and comes with three domains already modeled in the TSAL language. Two of these domains are described in Section 5.

The caveat with having a general, domain-independent novelty generator is ensuring only “high-quality” or “interesting” novelties are found. To alleviate this problem we introduce the approach in this paper where a human software developer works alongside the novelty generator. The effect is that the human is able to spend less time on the creative brainstorming phase and instead focus their energy on the “evaluation” phase. This also reduces the bias from only producing novelties along certain dimensions (such as the

²R and T refer to Wiggins’ Creative Systems Framework theory rule sets, otherwise ‘R’ and ‘T’ are simply variable names.

human getting fixated on novel kinds of objects instead of action/event properties).

Listing 1: Python example to run Novelty Generator

4 The Human-Dev ITL Process

The HITL method is time-efficient and removes human bias when developing novelties. Current method of novelty generation requires humans to perform brainstorming sessions to develop novel situations; and such an approach is highly time intensive, bias, and risks rework. Limitations of the current method are mitigated with the proposed HITL process that enables the user to generate, implement, and test novelties within a short time frame relative to the current typical method.

A general overview of our proposed novelty generation process is detailed below. Depending on the domain and expertise, the time to complete these steps varies.

Step 1: Construct domain specific TSAL files

The novelty generator requires a TSAL domain and problem file as input for novelty generation. Time to complete this initial step is variable depending on the HITL’s skill at creating an abstraction of the target domain and their ability to debug issues when passing the domain file through the TSAL interpreter.

Step 2: Run novelty generator using TSAL domain file

Once the TSAL domain file has been completed, the HITL will need to edit the novelty generator parameters to target a specific novelty type. We’ve found that a minimum of around 100 generated novelties is enough for further parsing.

Step 3: Identify possible novelties from generated files

The HITL must now manually look through the generated files and pick a novelty they think is viable. If the HITL sees fit, they may do some post-processing of certain actions or events by selecting noteworthy pre-conditions or events that can be moved from other generated files.

Step 4: Implement the novelty

The next step is to implement the novelty into the domain’s simulation space. Time to completion is variable depending on the modularity of the codebase and the complexity of the novelty. Initial implementation for specific novelty types or levels may take longer in order to set up the initial injection architecture, whereas subsequent implementations are relatively easier and quicker to complete.

Step 5: Test the novelty

The fifth step is to run experiments in order to test the viability of the novelty. These experiments consist of running a baseline agent through pre- and post-novelty scenarios while collecting data on the results. The time it takes to complete this step varies on the computation speed of the machine and the number of instances an experiment has. Viable novelties are determined by a signif-

icant increase or decrease in baseline agent performance when compared to a pre-novelty scenario.

Step 6: Revise the novelty

The final step is to revise novelty based on the experimental results. If the novelty does not have a significant increase or decrease in baseline agent performance when compared to a pre-novelty scenario, the HITL may revise the novelty generator parameters or parameters in the generated files to make the novelty viable. We consider this step optional as some novelties may never be viable, even with revisions.

5 Experimental Setup

To evaluate our proposed novelty generation process, a series of experiments was conducted in two domains to investigate the limitations and capabilities of the process. The two domains used to evaluate the HITL novelty generation process were (1) Monopoly and (2) VizDoom. The experimental setup for each domain are detailed below.

5.1 Monopoly Domain

The Monopoly domain is a python program that simulates the Monopoly board game that is developed under ISI’s GNOME³, an experimental platform created to evaluate the performance of multi-agent AI systems in the face of novelty. GNOME separates the development of AI game-playing agents from the simulator, allowing for the creation and injection of unanticipated novelty not disclosed to the agent.

Agent Setup Agents used in this domain can be found in the GNOME-p3 GitHub repository with the following modified rule set:

- Players don’t receive an extra turn on doubles
- Players automatically set free from jail after one turn

The experiments consisted of four total agents: one baseline reinforcement learning (RL) agent (Agent 1) and three identical background agents (Agent 2, Agent 3, and Agent 4). All three of these background agents follow the same decision-making processes in order to win the game. The background agents achieve victory through proper trades and property management in order to bankrupt the rest of the players. They are not biased against the player agent in the sense that they do not make decisions that are detrimental to ONLY the player agent.

Evaluation Setup To evaluate the HITL novelty generation process, each of the generated novelties were evaluated by their viability, or impact on the baseline agent. This was done by comparing win rates of agents in pre- and post-novelty tournaments. The average wins and standard deviation of the pre-novelty agent performance is shown in Table 1.

³<https://github.com/mayankkejriwal/GNOME-p3>

Table 1: Results of pre-novelty agent performance over 20 seeds of 10,000 games each.

	Agent 1	Default Background Agent
Average Wins	6943	1019
Raw Std Dev	45.66	30.07
% Win Rate Std Dev	0.66	2.96

The three levels of viability used to evaluate the novelties are defined below:

Low Viability: $0.66\% < \text{Agent 1} < 4\%$

Medium Viability: $4\% < \text{Agent 1} < 9\%$

High Viability: $\text{Agent 1} > 9\%$

The change in win rate used to assess viability of a novelty could either be an increase or decrease in win rate. In other words, a novelty that alters the baseline agent’s performance significantly enough could be considered viable.

For each experiment, we simulated three tournaments with 10,000 instances (games) per tournament. To keep results consistent, we used meta seeds 5, 10, 999 as inputs for the tournaments. The novelties developed using the HITL method for each novelty level are summarized in Table 2.

Table 2: Summary of developed agents used to verify and validate generated novelties of various levels.

ID	Description
0	Players gain \$1,000 when passing GO
1	Players lose \$500 when passing GO
2	Players can only move when dice values are identical
3	Players gain \$25 at the end of their turn
4	Player must be in jail to receive a property after trade is complete
5	Player must be in jail to receive cash after trade is complete
6	Property is automatically sold back to the bank after rent is paid for that property
7	Rent is not paid unless the owner of the space is in jail
8	The player cannot collect money if they are the last bidder to an auction

5.2 VizDoom Domain

VizDoom is an open-source project used as a research platform for RL using the popular first-person shooter game, Doom (1993), as the environment. This project allows researchers and developers to create and test learning and decision-making algorithms through the modular simulation

Table 3: Summary of developed agents used to verify and validate generated novelties of various levels.

ID	Description
0	Player loses 1 health point every 7 turns
1	Player loses one ammo clip every 40 turns
2	Player is turned 90 degrees every 3 turns
3	After rotating, the player loses 2 health points
4	After shifting positions, the player loses 1 health point
5	After rotating, the player is turned 15 degrees to the left

environment. We developed on top of Washington State University’s (WSU) VizDoom Novelty Generator GitHub⁴ to inject our custom novelties and conduct experiments on RL agents.

Agent Setup WSU provided their state-of-the-art (SOTA) player agent for testing, which we used as our baseline agent. While we are unable to go over the specifics of the baseline player agent’s RL logic, we are able provide a general overview of its functionality.

The agent has eight actions: forward, backward, left, right, turn left 45 degrees, turn right 45 degrees, shoot, and no action. For each turn, a feature vector of the current state of the simulation environment is sent to the agent; including general information about enemies, items, and the player agent. In response, the agent provides an action to be performed. Performance is defined as the amount of time left in the episode divided by the maximum time for the episode.

Training for the agent is done separately from testing. The user is free to indicate the number of training instances the agent should be trained on. When training is complete, the current model for the agent is saved in memory. We have pre-trained the SOTA agent for 100 pre-novelty instances before using it for novelty testing.

Evaluation Setup The VizDoom domain consists of a fork of the WSU VizDoom GitHub⁵; with a change of four total enemies and four of each item (health, ammo, trap) spawned for every instance. We conducted one experiment with 1000 instances for pre-novelty, and 1000 instances of each post-novelty. As a control and point of comparison, the SOTA agent was able to win 991 out of 1000 pre-novelty games averaging to a win-rate of 99.1 percent.

The novelties developed using the HITL method for each novelty level are summarized in Table 3.

⁴<https://github.com/holderlb/WSU-SAILON-NG/tree/master/WSU-Portable-Generator>

⁵<https://github.com/mbercasio94/WSU-SAILON-NG>

6 Results

6.1 Novelty Generation Setup

Independent of the TSAL file creation explained in step 1 of the HITL novelty creation process, the total time to find, implement, and test a new novelty ranges from 2 hours to 4 hours. Step 1 is removed from the time band since it varies depending on the user expertise and the target domain’s complexity.

6.2 Monopoly

The performance results of Agent 1 for all conducted novelties is shown in Table 4. The results indicate the majority of implemented novelties are of high viability, meaning a significant difference of $> 9\%$ in the agent performance. Table 4 contains a summary of the agent’s change in win rate over 10,000 games for each novelty.

Table 4: Viability of generated Monopoly novelties

ID	Post-Novelty Agent Absolute Win Rate			Viability
	Seed = 5	Seed = 10	Seed = 999	
0	-20.21%	-19.51%	-22.05%	High
1	5.14%	4.55%	4.04%	Medium
2	-28.61%	-27.11%	-26.90%	High
3	-26.71%	-26.04%	-28.01%	High
4	-20.09%	-19.35%	-18.97%	High
5	-15.59%	-15.17%	-16.31%	High
6	-28.06%	-30.33%	-28.42%	High
7	-99.90%	-99.87%	-99.94%	High
8	-96.48%	-95.99%	-96.25%	High
9	15.87%	14.75%	14.41%	High
10	-49.54%	-48.39%	-48.48%	High
11	-21.89%	-20.41%	-20.26%	High

6.3 VizDoom

The performance results of the SOTA agent for all conducted novelties is shown in Table 7. Since the average SOTA pre-novelty win rate was 99.1%, any novelties below a 90% win rate were considered high viability. Table 5 contains a summary of the agent’s win rate over 1,000 games for each novelty.

Table 5: Viability of generated VizDoom novelties.

ID	Win Rate	Viability
0	52.6%	High
1	46.0%	High
2	39.8%	High
3	0.4%	High
4	31.5%	High
5	0.0%	High

6.4 Discussion

During our experiments, most novelties using the HITL novelty generation process resulted in medium to high viability novelties, reinforcing our hypothesis that the generated novelties are typically sufficient to thoroughly test various agents. In the situation where a generated novelty was categorized as low viability, we were able to regenerate, re-implement, and retest new novelties in less than 4 hours. Our novel HITL method is an automated and time-efficient novelty generation process compared to current human-only novelty generation process.

7 Qualitative Review of Human Experience during Novelty Generation

In this section we go over a HITL’s initial thoughts for each step of the novelty generation process. This provides a look at the possible advantages and limitations of our current model in the perspective of a first-time user. Table 6 provides an overview of the time it takes to complete each step for both Monopoly and VizDoom domains.

Step 1: Construct domain specific TSAL files

(Difficulty - High) For those unfamiliar with PDDL or declarative languages in general, it might take several iterations and possible feedback from an expert in order to properly model the domain. Abstraction of a domain must account for certain nuances that might not be so obvious on the initial iteration.

In the case of the Monopoly domain, an extra turn from dice rolls was modelled in the TSAL file even though it was not part of the actual simulation logic, thus resulting in extra turn novelties that deviated too much from the original domain.

Step 2: Run novelty generator using TSAL domain file

(Difficulty - Low) Slight training by an expert or readme is required in order to know which lines of novelty generator code need to be adjusted in order to target specific novelty types.

Step 3: Identify possible novelties from generated files

(Difficulty - Low) As the HITL becomes more familiar with the domain, they should have a better understanding of the novelties that are likely to cause significant impact on agent performance. For example, the HITL may find a pattern where adjusting the preconditions of a specific event has higher likelihood of producing a viable novelty.

Step 4: Implement the novelty

(Difficulty - Medium) Creating novelty injection code depends on the modularity of the domain – how easy is it for a developer to adjust the rules and environment of the simulation space? In addition, difficulty depends on the type of novelty being implemented. For example, novelties that effect agent goals may require further implementation work when compared to a novelty that simply changes an event-based rule of the world. Similar to step 3, as the HITL familiarizes themselves with the code-base, implementation of further novelties should be relatively easier.

Table 6: Novelty Generator Experiment Results with HITL completion time

HITL Novelty Generation Step Description	Avg. Minutes to Complete Step	
	VizDoom	Monopoly
Step 1: Construct TSAL file specific to target domain	Variable	Variable
Step 2: Run novelty generator using TSAL domain file. This process includes editing of the novelty generator parameters to target specific novelty levels. Novelty file generation - target to generate at least 100 files.	5 minutes	5 minutes
Step 3: Identify possible novelties from generated files. Manually parse through generated files for novelties that may prove viable for the target novelty level. Post-processing of certain conditions and effects sometimes required.	30 minutes per novelty	30 minutes per novelty
Step 4: Implement the novelty. Time to implement a novelty depends on the level of that novelty. Initial implementation for a specific novelty level is slow (requires planning of software architecture). Subsequent implementations significantly faster for already created novelty levels.	45 minutes for the first novelty, 20 minutes for subsequent novelties	60 minutes for the first novelty, 20 minutes for subsequent novelties
Step 5: Test the novelty. Run experiment to test for novelty viability. Dependent on computation speed of the simulation machine.	30-60 minutes	1-2 hours
Step 6: Revise the novelty. Manually revise novelty values as needed. May also discard novelty if no changes work.	5 minutes	5 minutes

Step 5: Test the novelty

(Difficulty - Low) In terms of manual labor, a HITL will need to simply create a batch script in order to run experiments on the implemented novelties. Running experiments with more instances will take longer to complete, but results in a more accurate view of novelty viability. The threshold for the minimum change in performance required for a novelty to be deemed viable will depend on the judgement of the HITL.

Step 6: Revise the novelty

(Difficulty - Low) A HITL will need to simply change parameter(s) in the generated novelty file or novelty generator. As an example, for *Experiment 0* (Players gain \$1000 when passing GO), the value of \$1000 chosen by the novelty generator may not cause a significant change in agent performance. A user may opt to manually change the value to \$500 which may end up resulting in a more significant change in agent performance. Of course, some novelties will never be viable even with HITL revision. In that case, it is safe to discard the novelty and try a new one.

8 Conclusions and Future Work

We introduced a 6-step process by which a human with domain expertise can work alongside our open-source novelty generator to discover, implement, and score AI approaches. Our novelty generator is domain-independent, as demonstrated by our evaluation in two substantially different domains: Monopoly and VizDoom. Any environment that is defined in the Planning Domain Definition Language (PDDL) can be easily modified to work with our novelty

generator, since our TSAL language includes most PDDL constructs.

We hope to apply our novelty generator to more environments and develop a more complete testbed environment where agents can be tested directly against the state-transition system without requiring human developer effort to implement novelties in an external simulator. The TSAL language is capable and existing PDDL simulators exist, however at this time a TSAL simulator has not yet been built. A TSAL simulator would enable novelties produced by our method to be used in other frameworks like PDDL-gym (Silver and Chitnis 2020): a testbed for reinforcement learning approaches in PDDL domains.

Additionally, leveraging human guidance would be easier with a better user interface. Currently, our system writes novelties to a plain text file that a human can scan line-by-line. An improved user interface should offer features such as syntax highlighting and other visual indicators to aid the human in picking out good novelties, which we leave for future work.

While the results demonstrated our generator’s effectiveness in these areas, a significant limitation arises from its restricted scope. Our current model may not generalize well to other domains or interdisciplinary contexts. By limiting the generator’s capabilities to only two domains and two novelty levels, we may have inadvertently curtailed its potential to create groundbreaking ideas across a broader range of disciplines. Future research should focus on expanding the generator’s reach to encompass multiple domains, thus allowing for a more comprehensive exploration of novel concepts and enhancing its applicability in various fields.

Acknowledgements: This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001121C0236. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

References

2000. EGGG: Automated programming for game generation. *IBM Systems Journal*, 39(3.4): 782–794.
- Balloch, J.; Lin, Z.; Hussain, M.; Srinivas, A.; Wright, R.; Peng, X.; Kim, J.; and Riedl, M. 2022. Novgrid: A flexible grid world for evaluating agent response to novelty. *arXiv preprint arXiv:2203.12117*.
- Dannenhauer, D.; Reifsnnyder, N.; Regester, A.; and Molineaux, M. 2022. Transforming Environments to Evaluate Agent Adaptation.
- Goel, S.; Tatiya, G.; Scheutz, M.; and Sinapov, J. 2021. Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.
- Kejriwal, M.; and Thomas, S. 2021. A multi-agent simulator for generating novelty in monopoly. *Simulation Modelling Practice and Theory*, 112: 102364.
- Molineaux, M.; Dannenhauer, D.; and Kildebeck, E. 2023. A Framework for Characterizing Novel Environment Transformations in General Environments. *arXiv preprint arXiv:2305.04315*.
- PELL, B. 1992. METAGAME: a new challenge for games and learning. *Heuristic programming in artificial intelligence-the third computer olympiad*.
- Risi, S.; and Togelius, J. 2020. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8): 428–436.
- Senator, T. 2019. Science of AI and learning for openworld novelty (SAIL-ON). In *Proposers’ Day Meeting. DARPA: Arlington, VA*. https://www.darpa.mil/attachments/SAIL-ON_Proposers_Day_Distro_A_no_notes.pdf.
- Silver, T.; and Chitnis, R. 2020. PDDL Gym: Gym environments from PDDL problems. *arXiv preprint arXiv:2002.06432*.
- Smith, A. M.; and Mateas, M. 2010. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 273–280. IEEE.
- Wiggins, G. A. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7): 449–458.
- Xue, C.; Pinto, V.; Zhang, P.; Gamage, C.; Nikonova, E.; and Renz, J. 2022. Science Birds Novelty: An Open-world Learning Test-bed for Physics Domains. In *Proceedings of the AAAI Spring Symposium on Designing AI for Open-World Novelty, Palo Alto, CA, USA*, 21–23.

Novelty Analysis for ISI Novelty Generation Team

Matthew Molineaux – Parallax Advanced Research – matthew.molineaux@parallaxresearch.org
Prepared with support from PacMar Technologies

November 18, 2022



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

Analysis Purpose and Method

We are providing feedback to help your decision-making process on the SAIL-ON program as a TA2, not attempting to call your decisions into question. Our primary goal is to provide you with descriptions of the novelty dimensions and estimated difficulties of them, for you to consider; and to provide you with our assessments of how well they meet the novelty hierarchy level as we have formalized it in our theory (see Transformation and Hierarchy Formalization document).

For each novelty below, we describe the following elements:

- Target Level: The novelty hierarchy level you targeted
- Novelty Description: Your description of the novelty
- Assumptions: Assumptions we made about the novelty that may or may not be correct.
- Discussion: We use this section to provide context for how we look at the novelty and how we model it.
- Estimated T-SAL Environment Model Elements: Excerpts from a hypothetical T-SAL model of the modified and/or original environment that describe the novelty.
- Estimated T-SAL Transformation Sequence: Formal description of the novelty as a sequence of transformations between initial environment model and scenario generator to modified environment model and scenario generator; as described in the Transformation and Hierarchy Formalization document
- Novelty Dimensions: Assignment of novelty characteristics along eight environment transformation dimensions described in our Environment Transformation Framework theory, and explanation of each assignment
- Hierarchy Level Analysis: An analysis of whether the described novelty fits the definitions of hierarchy levels in the SAIL-ON program, based on our formal theory describing each.

Our assessments of novelty dimension values and hierarchy level analysis are based on the T-SAL language (itself primarily based on PDDL+) and transformations to environments described in that language. As such, the estimated transformation sequence is the formal basis for our determinations, and it's based on the estimated environment model elements required to describe a novelty. In many cases, multiple transformation sequences could create different environment models that would explain the observations generated equally well. When we see this as being possible, we have noted it.

The hierarchy level analysis for each novelty answers three questions:

1. Does the novelty, as described, clearly have the characteristics of the stated level as described in our theory? Might it be described at a lower one?
2. Does the novelty **also** have characteristics from a higher novelty level, which might make it more difficult than intended?

3. Is the novelty as simple as possible? This is related to PM Ted Senator’s request that only a “single” novelty be presented at each point. Our transformation sequences make clear that it is usually not possible to describe a novelty with only a single textual change to a model, so we ask instead whether a simpler transformation sequence can be constructed, by removing one or more existing elements, that still meets the requirements of the target novelty level. There may be circumstances in which meeting this criterion is impractical.

These questions have the following possible responses:

Q	Icon	Response	Meaning
1		<i>Analyzed level matches intended level</i>	Our theory of hierarchy levels treats the level of the novelty as the same one you provided.
		<i>Could be modelled at lower level</i>	We found a model for the novelty as described that matches the provided level in our theory of hierarchy levels, but we also found an alternative model at a lower novelty level. This might confuse some agents.
		<i>Analyzed level does not match intended level</i>	We did not find a model for the novelty that matches your provided level.
2		<i>Higher level novelties not present</i>	The transformation sequence found does not also meet the conditions for a higher level novelty than that stated.
		<i>Higher level novelties are present</i>	The transformation sequence found meets the conditions for a higher level novelty than that stated, in addition to conditions for a lower level.
3		<i>Simpler novelty not possible</i>	There is no way to remove a transformation from the T-SAL transformation sequence that models this novelty and arrive at a sequence from the described level.
		<i>Simpler novelty is possible</i>	One or more T-SAL transformations could be removed from this sequence, and the resulting sequence would still be a legal novelty at the intended level. (That novelty might be a nuisance novelty, though.)

In addition to the icon and response, we describe the rationale for each response, and provide a possible remedy, if you are interested in pursuing it, for “non-green” responses. There’s nothing binding about these, we’re just trying to provide the best feedback we can; please ignore whatever’s not helpful to you.

Monopoly Individual Level 6 Novelties

We've analyzed all the individual novelties that make up the composites sent to us in this section.

Target Level: 6 - Environment

Novelty Description:

player_starting_cash_novelty; 500

Each player starts the game with 500 dollars

Discussion:

In general, this is a simple novelty that can be given an arbitrary constant. The value of the constant does not affect properties of the novelty.

Estimated T-SAL Transformation Sequence:

ADDERELATIONGENERATOR(PAYER-CASH, ALLPERMUTATIONS([PLAYERS], CONSTANTFUNCTION(500))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: None: All transitions are identical to the original game.

Novelty Entry Determinism: None Novelty only modifies the starting state.

Novelty Obviousness: Entry Transition Observable The initial state and observation are different.

Novelty Observation Determinism: None All observations are the same as the original environment.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to initial observation change.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level *does not match* intended level: This is an agent novelty in our framework, as it modifies a property of a Monopoly player (agent). This is not an environmental novelty according to our framework; we consider environmental novelty to be due to change in distribution of environmental fluents (fluents outside agent control) or environmental events (events whose trigger conditions are outside agent control). This changes the distribution of a non-environmental fluent, PLAYER-CASH, which players can control through the choice to purchase, bid on properties, and trade.

Possible remedy: Environmental fluents in monopoly include rents, property costs, mortgage values, improvement costs, space positions, and component limits. Changes to these distributions are environmental.

- ✓ *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
- ✓ *Simpler novelty is not possible:* The transformation sequence consists of a single item, and is therefore irreducible.

Target Level: 6 - Environment

Novelty Description:

change_mortgage_percentage; 2.0

The player will be charged 300% of the original property mortgage amount when the player want to free the mortgaged property.

Discussion:

The specific percentage value is expected to change for different novelties; it shouldn't affect the novelty properties.

Assumed existing T-SAL action descriptions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
    (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
    (>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH))
    (NOT (SPACE-MORTGAGED ?SP))
  )
)
```

Estimated T-SAL description of novel actions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
    (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
(>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
    (>= (PLAYER-CASH ?P) (* 3 ?CASH))
  )
)
```

```
)
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 3 ?CASH))
    (NOT (SPACE-MORTGAGED ?SP))
  )
)
```

Estimated T-SAL Transformation Sequence:

```
REMOVEPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 3 ?CASH)))
REMOVEEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 3 ?CASH)))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Missing changes: Free_mortgage results in larger decrease in change value. Free_mortgage will also fail sometimes due to lack of funds.

Novelty Entry Determinism: Infrequent Nondeterministic: Entry into the novelty region happens as soon a player gains enough money to free an existing mortgage, which can happen through passing go, collecting rent, etc.

Novelty Obviousness: Novel Transition Discernible When freeing a mortgage, the failure or decreased amount of available cash is immediately evident.

Novelty Observation Determinism: Deterministic All Monopoly observations are deterministic.

Novelty Observation Localization: Nearby Novelty is not evident until player attempts to free a mortgage.

Novel State Observation Activity: N/A: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 **Analyzed level *does not match* intended level:** This is an action novelty as it modifies preconditions and effects of an action that can be chosen by a player. This is not an environmental novelty according to our framework, as it does not change any environmental transitions (those outside of an agent's control). Examples of environmental transitions in Monopoly are draws from chance and community chest, die rolls, transitions to jail, rent payments, and auction starts. **Possible Remedy:** In our framework, environment novelty should be triggered by an environmental condition, not a player decision. So paying an extra mortgage payment when the dice show a particular value, or when the player passes or lands at a particular point, for example, would be environmental.

- ✓ *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
- ✓ *Simpler novelty is not possible:* This is a relatively simple change, and removing elements of the transformation sequence result in irrelevant novelty or allow intentional entry to nonsensical states such as negative money.

Target Level: 6 - Environment

Novelty Description:

alternate_improve_property

Each player has to improve hotel before improving house

Discussion:

I read this as “hotels must be purchased before houses to improve a property”.

Assumptions:

- Player pays normal price for hotels.
- Hotels do not go away when a house is purchased.
- Rent for a property with hotel + houses is combined title rent.
- 2 hotels may not be purchased for a property.
- Due to component limitations (12 hotels in game), no more than 12 properties may be improved at one time.
- Maintaining even purchases across a color still applies, such that 3 hotels must be purchased on a 3-property color before the first house can be purchased.

Assumed existing T-SAL action descriptions

```
(:ACTION BUY_HOUSE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (EQ (MONOPOLY-MIN-HOUSES ?COLOR) ?MIN)
    (> (AVAILABLE-HOUSES) 0)
    (EQ (SPACE-HOUSES ?SP) ?MIN)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (+ ?MIN 1) ?NEW-HOUSES)
    (EQ (TITLE-HOUSES-RENT ?T ?NEW-HOUSES) ?NEW-RENT)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) ?NEW-HOUSES)
    (DECREASE (AVAILABLE-HOUSES) 1)
```

```

    (SET (SPACE-HOUSES ?SP) ?NEW-RENT)
  )
)

(:ACTION BUY_HOTEL
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4)
    (NOT (SPACE-HAS-HOTEL ?SP))
    (> (AVAILABLE-HOTELS) 0)
    (EQ (SPACE-HOUSES ?SP) 4)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) 0)
    (INCREASE (AVAILABLE-HOUSES) 4)
    (SPACE-HAS-HOTEL ?SP)
    (SET (SPACE-RENT ?SP) ?HOTEL-RENT)
  )
)

```

Estimated T-SAL description of novel actions

```

(:ACTION BUY_HOUSE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (EQ (MONOPOLY-MIN-HOUSES ?COLOR) ?MIN)
    (> (AVAILABLE-HOUSES) 0)
    (EQ (SPACE-HOUSES ?SP) ?MIN)
  )
)

```

```

(EQ (SPACE-TITLE ?SP) ?T)
(EQ (TITLE-HOUSE-COST ?T) ?COST)
(>= (PLAYER-CASH ?P) ?COST)
(EQ (+ ?MIN 1) ?NEW-HOUSES)
(EQ (TITLE-HOUSES-RENT ?T ?NEW-HOUSES) ?NEW-RENT)
(EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
(FORALL (?SAME-COLOR-SPACE - SPACE)
  (EQ (SPACE-COLOR ?SAME-COLOR-SPACE) ?COLOR)
  (SPACE-HAS-HOTEL ?SAME-COLOR-SPACE)
)
)
)
:EFFECT
(AND
  (DECREASE (PLAYER-CASH ?P) ?COST)
  (SET (SPACE-HOUSES ?SP) ?NEW-HOUSES)
  (DECREASE (AVAILABLE-HOUSES) 1)
  (SET (SPACE-HOUSES ?SP) (+ ?NEW-RENT ?HOTEL-RENT))
)
)

(:ACTION BUY_HOTEL
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
(>= (MONOPOLY-MIN-HOUSES ?COLOR) 4)
    (> (AVAILABLE-HOTELS) 0)
(EQ (SPACE-HOUSES ?SP) 4)
    (EQ (SPACE-HOUSES ?SP) 0)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
  )
)
:EFFECT
(AND
  (DECREASE (PLAYER-CASH ?P) ?COST)
  (SET (SPACE-HOUSES ?SP) 0)
  (INCREASE (AVAILABLE-HOUSES) 4)
  (SPACE-HAS-HOTEL ?SP)
)

```

```
(SET (SPACE-RENT ?SP) ?HOTEL-RENT)
)
)
```

Estimated T-SAL Transformation Sequence:

```
ADDPRECONDITION(BUY_HOUSE, (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT))
ADDPRECONDITION(BUY_HOUSE, (FORALL (?SAME-COLOR-SPACE — SPACE) (EQ (SPACE-COLOR ?SAME-COLOR-SPACE) ?COLOR) (SPACE-HAS-HOTEL ?SAME-COLOR-SPACE))))
REMOVEEFFECT(BUY_HOUSE, (SET (SPACE-HOUSES ?SP) ?NEW-RENT))
ADDEFFECT(BUY_HOUSE, (SET (SPACE-HOUSES ?SP) (+ ?NEW-RENT ?HOTEL-RENT)))
REMOVEPRECONDITION(BUY_HOTEL, (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4))
REMOVEPRECONDITION(BUY_HOTEL, (EQ (SPACE-HOUSES ?SP) 4))
ADDPRECONDITION(BUY_HOTEL, (EQ (SPACE-HOUSES ?SP) 0))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change, Added Change: Buying a house doesn't work in states with monopolies and no hotels; buying a house does work in states with hotels

Novelty Entry Determinism: Agent-Caused/Infrequent nondeterministic Novelty entry occurs when a player becomes able to purchase a hotel because either the player had enough cash left over after completing a monopoly (agent-caused), or the player recently gained enough cash to purchase a house on an existing monopoly (infrequent nondeterministic).

Novelty Obviousness: Novel Transition Discernible A failed BUY_HOTEL action is immediately discernible to the player that tried.

Novelty Observation Determinism: Deterministic All observation are deterministic in Monopoly.

Novelty Observation Localization: Nearby Agent will not observe novelty without trying to buy a house or hotel, at which point a novel transition occurs first.

Novel State Observation Activity: N/A: No new states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



*Analyzed level **does not match** intended level:* This is an action novelty according to our framework, as it changes the preconditions of player actions. **Possible remedy:** Changes to houses and hotels triggered by the environment, such as losing them in response to a community chest card draw, would be environment events.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.

 **Simpler novelty *is possible*:** This appears somewhat complex from a transformation sequence perspective. It's possible to have a much simpler sequence by simply allowing hotels to be bought at any time rather than requiring them to be bought first.

Target Level: 6 - Environment

Novelty Description:

granularity_novelty1; Tennessee Avenue, 20

There are 20 duplicated Tennessee Avenue appearing after original Tennessee Avenue on the board

Assumptions:

This replaces every space between Tennessee and Go (New York Avenue through Boardwalk, including chance, go to jail, and free parking)

The person who owns Tennessee avenue title collects rent on all of the Tennessee avenue spots.

Tennessee Avenue spaces cannot be improved because New York Avenue can't be purchased and will never be auctioned off.

(For other granularity novelties, we assume that if a monopoly can be gained every space is improved separately and counts as part of the color group so must be improved evenly.)

Board property/title relationships are observed in initial observation.

Estimated T-SAL Transformation Sequence:

ADDVALUEGENERATOR(TENNSPACES, UNIFORMINTEGERDISTRIBUTION(20, 39))

ADDOBJECTGENERATOR(EXTRATENNS, OBJECTLIST(20, "PROP-4-2-"))

ADDERELATIONGENERATOR(POSITION-SPACE, ALLPERMUTATIONS([TENNSPACES], EXTRATENNS)

ADDERELATIONGENERATOR(SPACE-NUM, ALLPERMUTATIONS([EXTRATENNS], TENNSPACES)

ADDERELATIONGENERATOR(SPACE-TYPE, ALLPERMUTATIONS([EXTRATENNS], CONSTANTFUNCTION([PROPERTY]))

ADDERELATIONGENERATOR(SPACE-TITLE, ALLPERMUTATIONS([EXTRATENNS], TITLE-4-2)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Added Changes: Collection of rent happens in many unusual states to player with Tennessee Avenue

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Changed properties create new observations of board properties.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 *Analyzed level **does not match** intended level:* According to our framework, this is a relation novelty; the relationship between board spaces and titles is changed, such that many board spaces are related to a different title (Tennessee Avenue's). **Possible remedy:** If all states after Tennessee Avenue triggered a separate event, such as a money loss, it would meet our framework's definition of an environmental novelty.

 *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.

 *Simpler novelty is not possible:* All supporting generators are required to meet game state invariants.

Target Level: 6 - Environment

Novelty Description:

color_reordering; Pennsylvania Avenue', 'Pacific Avenue', 'North Carolina Avenue'
Rearrange the sequence of Green color set from: Pacific Avenue(31) -> North Carolina Avenue (32) -> ... -> Pennsylvania Avenue (34) to Pennsylvania Avenue(31) -> Pacific Avenue(32) -> ... -> North Carolina Avenue (34)

Discussion:

I read this as swapping the positions of these properties on the board without changing their values.

Assumptions:

This overrides the property duplication (granularity_novelty) when it occurs after it in a list, resulting in the rearranged properties being preserved (though out of order).

Property positions sent in initial observation.

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(POSITION-SPACE, [31], PROP-7-3)

ADDFLUENTVALUE(POSITION-SPACE, [32], PROP-7-1)

ADDFLUENTVALUE(POSITION-SPACE, [34], PROP-7-2)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value: Rent values change for these three properties, primarily.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level does not match intended level: This is an object novelty according to our framework, as only specific relation values are affected. **Possible Remedy:** If the

property switch occurs at a random time during the game, instead of the initial state, or in response to an environmental trigger such as a special dice roll, this becomes an environmental event.

-  *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
-  *Simpler novelty **is technically possible**:* Changing only a single property or pair of properties would be simpler, but not significantly so.

Target Level: 6 - Environment

Novelty Description:

card_novelty; {'bank_error': 50}, {'general_repairs': 50}

Increase 50 more community cards "bank_error" and 50 more chance cards "general_repairs" on the board

Assumptions:

Assuming that chance and community chest decks are implemented by a random number generator at draw time, rather than having a real shuffle and discard pile, which can be remembered.

Discussion:

Every Community Chest and Chance card is a separate event, and there's no simple way of replacing them en masse, so this is a relatively complicated change.

Assumed existing T-SAL action descriptions

```
(:EVENT VISIT-CHANCE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TYPE ?SPACE) CHANCE)
    (ASSIGN ?CHANCE-CARD (:UNIFORM 1 17))
  )
:EFFECT
  (AND
    (VISIT-CHANCE ?P)
    (SET (CHANCE-CARD) ?CHANCE-CARD)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
(:EVENT CHEST-BANK-ERROR
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHEST-CARD) 4)
    (EQ (CURRENT-PLAYER) ?P)
  )
)
```

```

:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 200)
  )
)
(:EVENT VISIT-COMMUNITY-CHEST
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TYPE ?SPACE) CHEST)
    (ASSIGN ?CHEST-CARD (:UNIFORM 1 17))
  )
:EFFECT
  (AND
    (VISIT-CHEST ?P)
    (SET (CHEST-CARD) ?CHEST-CARD)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
(:EVENT CHANCE-GENERAL-REPAIRS
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 8)
  )
:EFFECT
  (AND
    (NEED-MAINTENANCE 25 100)
  )
)

```

Estimated T-SAL description of novel actions

```

(:EVENT VISIT-CHANCE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
  )
)

```

```
(EQ (SPACE-TYPE ?SPACE) CHANCE)
(ASSIGN ?CHANCE-CARD (:UNIFORM 1 67))
)
:EFFECT
(AND
  (VISIT-CHANCE ?P)
  (SET (CHANCE-CARD) ?CHANCE-CARD)
  (SET (TURN-STAGE) POST_ROLL)
)
)
(:EVENT CHEST-BANK-ERROR-EXTRA
:PARAMETERS ()
:PRECONDITION
  (AND
    (> (CHEST-CARD) 17)
    (EQ (CURRENT-PLAYER) ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 200)
  )
)
(:EVENT VISIT-COMMUNITY-CHEST
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TYPE ?SPACE) CHEST)
    (ASSIGN ?CHEST-CARD (:UNIFORM 1 67))
  )
:EFFECT
  (AND
    (VISIT-CHEST ?P)
    (SET (CHEST-CARD) ?CHEST-CARD)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
(:EVENT CHANCE-GENERAL-REPAIRS-EXTRA
:PARAMETERS ()
:PRECONDITION
```

```
(AND
  (> (CHANCE-CARD) 17)
)
:EFFECT
(AND
  (NEED-MAINTENANCE 25 100)
)
)
```

Estimated T-SAL Transformation Sequence:

```
REMOVEPRECONDITION(VISIT-CHANCE, (ASSIGN ?CHANCE-CARD (:UNIFORM 1 17)))
ADDPRECONDITION(VISIT-CHANCE, (ASSIGN ?CHANCE-CARD (:UNIFORM 1 67)))
REMOVEPRECONDITION(VISIT-CHEST, (ASSIGN ?CHEST-CARD (:UNIFORM 1 17)))
ADDPRECONDITION(VISIT-CHEST, (ASSIGN ?CHEST-CARD (:UNIFORM 1 67)))
ADDEVENT(CHANCE-GENERAL-REPAIRS-EXTRA)
ADDPRECONDITION(CHANCE-GENERAL-REPAIRS-EXTRA, (> (CHANCE-CARD) 17))
ADDEFFECT(CHANCE-GENERAL-REPAIRS-EXTRA, (NEED-MAINTENANCE 25 100))
ADDEVENT(CHEST-BANK-ERROR-EXTRA)
ADDPRECONDITION(CHEST-BANK-ERROR-EXTRA, (> (CHEST-CARD) 17))
ADDEFFECT(CHEST-BANK-ERROR-EXTRA, (INCREASE (PLAYER-CASH ?P) 200))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Modified Change Rate: Visiting community chest and chance have changed rate of cash increase and maintenance demands, respectively

Novelty Entry Determinism: Infrequent Nondeterministic Entry occurs when an agent moves to a chance or community chest space, on a nondeterministic roll of the dice.

Novelty Obviousness: Statistical Novelty Only If on every visit to chance and community chest, any card can be drawn, the only difference is that two particular cards are suddenly much more frequent. If a discard pile exists, the second draw can be discerned as different given memory; under this differing assumption, obviousness is **Novelty Region Inferable**.

Novelty Observation Determinism: None All modified environment observations are possible in original environment.

Novelty Observation Localization: Nearby Agent must recognize novelty after it is affected.

Novel State Observation Activity: None: No new states exist, only transitions are affected.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This is an environmental novelty, as it modifies environmental events (new chances of visiting cards).



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.

 *Simpler novelty is possible*: Modifying both decks is unnecessary, so a simpler novelty is possible. **Possible remedy**: A simpler novelty modifies only one deck; the simplest version would remove a single card's effect or adds to it without changing draw probability.

Target Level: 6 - Environment

Novelty Description:

global_reordering

Rearrange the location sequence to: ['Virginia Avenue', 'Go', 'Community Chest', 'Mediterranean Avenue', 'Oriental Avenue', 'Income Tax', 'Reading Railroad', 'Baltic Avenue', 'Vermont Avenue', 'Connecticut Avenue', 'In Jail/Just Visiting', 'Chance', 'St. Charles Place', 'Electric Company', 'States Avenue', 'Pennsylvania Railroad', 'St. James Place', 'Community Chest', 'Tennessee Avenue', 'New York Avenue', 'Free Parking', 'Kentucky Avenue', 'Chance', 'Indiana Avenue', 'Illinois Avenue', 'B&O Railroad', 'Atlantic Avenue', 'Ventnor Avenue', 'Water Works', 'Marvin Gardens', 'Go to Jail', 'Pacific Avenue', 'North Carolina Avenue', 'Community Chest', 'Pennsylvania Avenue', 'Short Line', 'Chance', 'Park Place', 'Luxury Tax', 'Boardwalk']. All players start at location Virginia Avenue.

Assumptions:

Property locations are given in initial observation.

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(POSITION-SPACE, [0], PROP-3-3)
ADDFLUENTVALUE(POSITION-SPACE, [1], CORNER-GO)
ADDFLUENTVALUE(POSITION-SPACE, [2], CHEST-1)
...

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Missing Change, Added Changes:

Rolls result in different changes to rents and taxes, go event does not occur at different position, does occur at new position, etc.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to initial observation change.

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 **Analyzed level *does not match* intended level:** This is an object novelty in our framework, as it modifies where each original board space (an object) is encountered.

Possible remedy: To get the flavor of space warping without a constant change to specific objects, you might consider altering die rolls to work differently; a wildly different distribution, or teleporting to a different space every time you roll doubles, would be an environmental event in our framework.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



*Simpler novelty **is possible**:* Modifying smaller numbers of spaces would be simpler, as done in granularity_novelty, which seems like a less chaotic version of the same novelty.

Target Level: 6 - Environment

Novelty Description:

mortgage_novelty; real_estate, 60

Change the mortgage amount of all real estate properties to 60 dollars

Assumptions:

Mortgage values are not observed initially, only when the player mortgages a property.

Estimated T-SAL Transformation Sequence:

ADDERELATIONGENERATOR(TITLE-MORTGAGE-VALUE, ALLPERMUTATIONS([TITLES], CONSTANTFUNCTION(60)))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value: mortgaging gives different value.

Novelty Entry Determinism: Agent-Caused Novelty region is entered when a player purchases a property, and can immediately mortgage it if desired.

Novelty Obviousness: Novel Transition Discernible New cash change is discernible when a player mortgages a property.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent observes novelty after mortgaging a property.

Novel State Observation Activity: Passive States are not new.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 *Analyzed level **does not match** intended level:* This is an object novelty, as it modifies the properties of objects (titles or spaces).

 *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.

 *Simpler novelty is not possible:* Changing the mortgage value relation to a constant function is atomic.

Target Level: 6 - Environment

Novelty Description:

die_novelty; 1, [1, 2]

Only one dice with number [1, 2] on the board. The number [1, 2] occurrence probability distribution is [1/2, 1/2].

Assumption:

Dice are observable independently, not just their sum.
Agent doesn't see the dice before rolling.

Discussion:

Die_novelty_type novelties are not treated separately, they appear to have essentially identical properties.

Assumed existing T-SAL action descriptions

```
(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (:UNIFORM 1 6))
  (SET (DIE2-VALUE) (:UNIFORM 1 6))
  (SET (TURN-STAGE) MOVE))
)
```

Estimated T-SAL description of novel actions

```
(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (:UNIFORM 1 2))
  (SET (DIE2-VALUE) (:UNIFORM 1 6))
  (SET (DIE2-VALUE) 0)
  (SET (TURN-STAGE) MOVE))
)
```

Assumed existing T-SAL action descriptions

```
REMOVEEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (:UNIFORM 1 6)))
REMOVEEFFECT(DIE-ROLL, (SET (DIE2-VALUE) (:UNIFORM 1 6)))
ADDEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (:UNIFORM 1 2)))
ADDEFFECT(DIE-ROLL, (SET (DIE2-VALUE) 0))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Modified Change Rate, Missing Change: Different die rolls are different rates of outcomes such as distance and doubles. Doubles can no longer happen, which is a missing change.

Novelty Entry Determinism: Agent-Caused Entering the die rolling state is caused by another player ending their turn.

Novelty Obviousness: Alien Observation Rolling only one die is immediately observable, and a sum of 1 is a new possibility.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Dice roll occurs before new values are observed.

Novel State Observation Activity: None: No new states exist.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.



Hierarchy Level Analysis

Analyzed level matches intended level: This is an environmental novelty, as it modifies environmental events (die rolls).



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is not possible: Despite the sequence shown, this is really a change to a single distribution of outcomes, so it's reasonable to say this is a simplest possible novelty.

Target Level: 6 - Environment

Novelty Description:

price_novelty; 50, 'real_estate', 'States Avenue'
Change the price of States Avenue from 140 to 50 dollars

Assumptions:

Discussion:

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(TITLE-COST, [TITLE-3-2], 50)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Modified Change Rate: different die rolls; rolls result in different rents; purchasing states avenue gives different decrease in cash.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 *Analyzed level matches intended level:* This is an object novelty, as it modifies a property of one object (i.e., States Avenue). **Possible Remedy:** To be an environmental novelty, this would have to impact the task space as a triggered condition; for example, the cost might change the first time someone landed on States Avenue or increase every time doubles are rolled.

 *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.

 *Simpler novelty is not possible:* This is an atomic change.

Target Level: 6 - Environment

Novelty Description:

assign_pick_card_func

Applying card effect on all the players when one player pick up a community or chance card.

Assumptions:

The “pick-up” effect of a get out of jail free card is getting the card to use later, not getting you out of jail

Discussion:

Every Community Chest and Chance card is a separate event, and there’s no simple way of replacing them en masse, so this is a relatively complicated change.

Assumed existing T-SAL action descriptions

```
(:EVENT CHANCE-GO-TO-JAIL
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 1)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (SET (PLAYER-STATUS ?P) CAUGHT)
  )
)

(:EVENT CHANCE-GET-OUT-OF-JAIL-FREE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 2)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-GET-OUT-OF-JAIL-FREES ?P) 1)
  )
)
```

Estimated T-SAL description of novel actions

```
(:EVENT CHANCE-GO-TO-JAIL
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 1)
    (VISITING-CHANCE ?ANYONE)
    (IS-PLAYER ?P)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (SET (PLAYER-STATUS ?P) CAUGHT)
  )
)
```

```
(:EVENT CHANCE-GET-OUT-OF-JAIL-FREE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 2)
    (VISITING-CHANCE ?ANYONE)
    (IS-PLAYER ?P)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-GET-OUT-OF-JAIL-FREES ?P) 1)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDPRECONDITION(CHANCE-GO-TO-JAIL, (VISITING-CHANCE ?ANYONE))
ADDPRECONDITION(CHANCE-GO-TO-JAIL, (IS-PLAYER ?P))
REMOVEPRECONDITION(CHANCE-GO-TO-JAIL, (VISITING-CHANCE ?P))
ADDPRECONDITION(CHANCE-GET-OUT-OF-JAIL-FREE, (VISITING-CHANCE ?ANYONE))
ADDPRECONDITION(CHANCE-GET-OUT-OF-JAIL-FREE, (IS-PLAYER ?P))
REMOVEPRECONDITION(CHANCE-GET-OUT-OF-JAIL-FREE, (VISITING-CHANCE ?P))
... (All other chance and chest cards changed similarly)
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Added Changes: Values change in modified domain when *other* players go to chance or community chest

Novelty Entry Determinism: Agent Caused Novelty occurs when a player rolls and lands on chance or community chest; entry occurs when a player passes, and another player can roll.

Novelty Obviousness: Novel Transition Observable When another player lands on community chest or chance, extra changes observed immediately.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Novelty is not observed until new effects occur.

Novel State Observation Activity: None All states are the same.

Novelty Frequency: Less Frequent than Observation Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is an environmental novelty, as it modifies the effect of randomly drawn cards.
- ✓ *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
- ✗ *Simpler novelty **is possible**:* Each card has different effects that have to be applied to different players. In our formalism, there's no way to change from one to many globally, so this is a very complex change compared to modifying a single card.

Target Level: 6 - Environment

Novelty Description:

percent_of_total_mortgage_outstanding; 2.0

The player will be charged 300% of the total debt of all mortgaged properties when the player want to free one mortgaged property.

Assumptions:

Cost based on all properties, not all owned by current player

Assumed existing T-SAL action descriptions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
    (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
    (>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH))
    (NOT (SPACE-MORTGAGED ?SP))
  )
)
```

Estimated T-SAL description of novel actions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
(EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
(>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
  )
)
```



```

(SUM ?TOTAL (?MTG-SP - SPACE)
  (AND (SPACE-MORTGAGED ?MTG-SP) (EQ (SPACE-TITLE ?MTG-SP) ?MTG-TITLE)
    (TITLE-MORTGAGE-VALUE ?MTG-TITLE)
  )
  (>= (PLAYER-CASH ?P) (* 3 ?TOTAL))
)
:EFFECT
(AND
  (DECREASE (PLAYER-CASH ?P) (* 3 ?TOTAL))
  (NOT (SPACE-MORTGAGED ?SP))
)
)

```

Estimated T-SAL Transformation Sequence:

```

REMOVEPRECONDITION(FREE_MORTGAGE, (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH))
REMOVEPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDPRECONDITION(FREE_MORTGAGE, (SUM ?TOTAL (?MTG-SP - SPACE) (AND (SPACE-MORTGAGED ?MTG-SP)
  (EQ (SPACE-TITLE ?MTG-SP) ?MTG-TITLE) (TITLE-MORTGAGE-VALUE ?MTG-TITLE))))
ADDPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 3 ?TOTAL)))
REMOVEEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 3 ?TOTAL)))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change, Added Changes: free_mortgage results in larger decrease in change value; free_mortgage doesn't work sometimes

Novelty Entry Determinism: Agent-Caused Mortgaging a property leads to the agent being able to free it, this is an agent decision.

Novelty Obviousness: Novel Transition Discernible Freeing a mortgage will cause different results immediately.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until someone attempts to free a mortgage

Novel State Observation Activity: None: All states are the same

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 **Analyzed level matches intended level:** This is an action novelty, as it modifies the effects of a player selected action. **Possible remedy:** If a player was instead required to pay a percentage of the mortgage total upon passing go, for example, this would be outside the player's control and therefore an environmental novelty.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



*Simpler novelty **is possible**:* A simpler novelty could be made by considering the value of a single property rather than all of them.

Target Level: 6 - Environment

Novelty Description:

price_per_house_novelty; 'real_estate', 'Boardwalk', 10
Change the price of the house of Boardwalk from 200 to 10 dollars

Assumptions:

Price of house on Boardwalk is given in initial observation

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(TITLE-HOUSE-COST, [TITLE-8-2], 10)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value Buying house on Boardwalk decreases funds less

Novelty Entry Determinism: None Novelty exists immediately since the price of a Boardwalk house changes

Novelty Obviousness: Alien Observation Price of house on Boardwalk changes initial observation out of distribution.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to house price

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



*Analyzed level **does not match** intended level:* This is an object novelty, as it modifies a property of the Boardwalk object.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is not possible: This is an atomic novelty and can not be made simpler.

Target Level: 6 - Environment

Novelty Description:

bank_attribute_novelty; 'total_houses', 9
Change the available houses in bank from 32 to 9

Assumptions:

Available houses is public information

Discussion:

ADDERELATIONGENERATOR(AVAILABLE-HOUSES, ALLPERMUTATIONS([], CONSTANTFUNCTION(9))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change: Buying a house fails when bank runs out, so there is a missing change when buy_house stops having its preconditions met.

Novelty Entry Determinism: None Novelty exists immediately since the # available houses changes.

Novelty Obviousness: Alien Observation # available houses changes observation immediately.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to # available houses.

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level *does not match* intended level: This technically does not meet the definition of any novelty level, since only initial state is changed and houses are not a property of any object. It's closest to action novelty, since this affects the rules only through the ability to buy a house. **Possible remedy:** If instead of starting with fewer houses, some environmental feature caused them to be lost – like 1 per pass of go or something – that would be environmental.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is not possible: This novelty is as simple as possible.

Target Level: 6 - Environment

Novelty Description:

card_amount_novelty; {'doctor_fee': 100}, {'bank_dividend': 100}

The player will receive 100 dollars instead of paying 50 dollars when pick up community card "doctor_fee". The player will receive 100 dollars instead of receiving 50 dollars when pick up chance card "bank_dividend".

Discussion:

This likely meant “pay 100 dollars” for “doctor fee”, encoding it as such.

Assumed existing T-SAL action descriptions

```
(:EVENT CHANCE-BANK-DIVIDEND
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 15)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 50)
  )
)
(:EVENT CHEST-DOCTOR-FEE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHEST-CARD) 5)
    (VISITING-CHEST ?P)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) 50)
  )
)
```

Estimated T-SAL description of novel actions

```
(:EVENT CHANCE-BANK-DIVIDEND
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 15)
    (VISITING-CHANCE ?P)
  )
)
```

```

:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 100)
  )
)
(:EVENT CHEST-DOCTOR-FEE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHEST-CARD) 5)
    (VISITING-CHEST ?P)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) 100)
  )
)

```

Estimated T-SAL Transformation Sequence:

```

REMOVEEFFECT(CHANCE-BANK-DIVIDEND, (INCREASE (PLAYER-CASH ?P) 50))
ADDEFFECT(CHANCE-BANK-DIVIDEND, (INCREASE (PLAYER-CASH ?P) 100))
REMOVEEFFECT(CHEST-DOCTOR-FEE, (DECREASE (PLAYER-CASH ?P) 50))
ADDEFFECT(CHEST-DOCTOR-FEE, (DECREASE (PLAYER-CASH ?P) 100))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Value Some community chest and chance changes are increased in size; purchasing houses has an increase in the rent value; paying rent on Atlantic Avenue / Marvin Gardens increases and decreases, respectively

Novelty Entry Determinism: None Novelty exists immediately since the yellow properties are switched.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to yellow properties.

Novel State Observation Activity: Active: Agent must attempt to buy a hotel to observe a hotel on one property of monopoly, nothing on others.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is an environmental novelty, as it modifies the effects of environmental events (chest-doctor-fee and chance-bank-dividend).
- ✓ *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
- ✗ *Simpler novelty **is possible**:* This would be an environmental novelty by just modifying an effect of one card, so two is unnecessary.

Target Level: 6 - Environment

Novelty Description:

tax_novelty; 'Income Tax', 400

Change the tax amount at location 'Income Tax' from 200 to 400

Assumption:

Income tax amount is not exposed in initial observation

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(SPACE-RENT, [INCOME-TAX], 400)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Value Income tax rent causes larger cash decrease

Novelty Entry Determinism: Agent-Caused When any players passes the turn, the next player may be within the novelty area because they could land on income tax.

Novelty Obviousness: Novel Transition Observable Greater money loss should be observed immediately.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Some player must pay the tax before it can be observed.

Novel State Observation Activity: None: No states are novel.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Could be modelled at lower level: This is most easily treated as an object novelty, as the tax is a fixed amount that the visitor has to pay, just like rents on properties. It could be considered as an environment novelty if a special environment event just for charging income tax existed. **Possible Remedy:** This would more clearly be an environment novelty if a tax was charged based on some environmental factor other than a particular board location, such as a dice value.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is not possible: There's only one transformation in the sequence, so no novelty is simpler.

Target Level: 6 - Environment

Novelty Description:

map_property_to_color; Atlantic Avenue', 'Blue'
Change the color of Atlantic Avenue from Yellow to Blue

Assumptions:

Light blue (e.g. Oriental), not dark blue (e.g., Boardwalk).

Monopolizing blue now requires 4 properties, monopolizing yellow requires 2.

It's observable when a monopoly is obtained.

Colors of properties are in initial observation.

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(SPACE-COLOR, [PROP-6-1], LIGHT_BLUE)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Changes, Added Changes Monopolies for yellow now happen more easily and light blue less easily; so purchasing the second for yellow adds a monopoly change, and purchasing the third for light blue does not.

Novelty Entry Determinism: None Novelty is clear from initial observation.

Novelty Obviousness: Alien Observation A different color is outside the distribution of the original environment.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Novelty is observed before any action is taken.

Novel State Observation Activity: Passive: All states are novel, and observably so.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



*Analyzed level **does not match** intended level.* In our framework, this is an object novelty, as only a property of a single object changes. **Possible Remedy:** For this to be environmental, something has to change during play. For instance, the color of a property could change when a player on that property rolled doubles.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is not possible: This novelty sequence is atomic.

Target Level: 6 - Environment

Novelty Description:

exchange_colors_between_properties; 'Vermont Avenue', 'Pacific Avenue'

Change the color of Vermont Avenue from SkyBlue to Green. Change the color of Pacific Avenue from Green to SkyBlue.

Assumptions:

It's observable when a monopoly is obtained.

Colors of properties are in initial observation.

Discussion:

This is the same effect as two map_color_to_property novelties together.

Estimated T-SAL Transformation Sequence:

ADDFLUENTVALUE(SPACE-COLOR, [PROP-2-2], GREEN)

ADDFLUENTVALUE(SPACE-COLOR, [PROP-7-1], SKY_BLUE)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Changes, Added Changes Monopolies for green and sky blue now happen under different circumstances.

Novelty Entry Determinism: None Novelty is clear from initial observation.

Novelty Obviousness: Alien Observation A different color is outside the distribution of the original environment.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Novelty is observed before any action is taken.

Novel State Observation Activity: Passive: All states are novel, and observably so.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 *Analyzed level **does not match** intended level.* In our framework, this is an object novelty, as only a property of a single object changes. **Possible Remedy:** For this to be environmental, something has to change during play. For instance, the colors of 2 properties could be exchanged when a player on one property rolled doubles.

 *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.

 *Simpler novelty **is possible**:* Another novelty is monotonically simpler.

Target Level: 6 - Environment

Novelty Description:

go_increment_novelty; 90

Change the receiving cash from 200 to 90 dollars when player passing the Go

Assumed existing T-SAL action descriptions

```
(:EVENT COLLECT-SALARY
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (TURN-STAGE) MOVE)
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (DIE1-VALUE) ?DIE1)
    (EQ (DIE2-VALUE) ?DIE2)
    (EQ (PLAYER-STATUS ?P) FREE)
    (EVAL (> (+ ?DIE1 ?DIE2) (PLAYER-POSITION ?P))))
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 200)
    (SET (TURN-STAGE) VISIT)
  )
)
```

Estimated T-SAL description of novel actions

```
(:EVENT COLLECT-SALARY
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (TURN-STAGE) MOVE)
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (DIE1-VALUE) ?DIE1)
    (EQ (DIE2-VALUE) ?DIE2)
    (EQ (PLAYER-STATUS ?P) FREE)
    (EVAL (> (+ ?DIE1 ?DIE2) (PLAYER-POSITION ?P))))
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 90)
    (SET (TURN-STAGE) VISIT)
  )
)
```

Estimated T-SAL Transformation Sequence:

REMOVEEFFECT(COLLECT-SALARY, (INCREASE (PLAYER-CASH ?P) 200))

ADDEFFECT(COLLECT-SALARY, (INCREASE (PLAYER-CASH ?P) 90))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Value Amount of money for passing go decreases.

Novelty Entry Determinism: Agent-Caused Entry occurs when a player passes and the next player could roll past go

Novelty Obviousness: Novel Transition Observable Changed salary collection is obvious.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Novelty observed only after salary is collected.

Novel State Observation Activity: None No states are new

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This is an environmental novelty, as it modifies the effects of an environmental event (collect-salary).



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



*Simpler novelty **is not possible**:* A simpler transformation sequence wouldn't be meaningfully different.

Monopoly Combined Level 6 Novelties

Before determining that individual novelties would be a more useful analysis, we created a number of analyses of composite novelties. Full analyses for rows 1-5 and 12 are given below. As a general rule, a composite novelty could always be simpler and is a member of the highest novelty level of its constituent novelties. Since the individual novelties tend to override one another in composites, these analyses remove detail found in the analyses of individual novelties. For a more in-depth analysis, see the very end.

Target Level: 6 - Environment

Novelty Description:

environments_1

1. Each player starts the game with 500 dollars
2. The player will be charged 300% of the original property mortgage amount when the player want to free the mortgaged property.
3. Each player has to improve hotel before improving house
4. There are 20 duplicated Tennessee Avenue appearing after original Tennessee Avenue on the board
5. Rearrange the sequence of Green color set from: Pacific Avenue(31) -> North Carolina Avenue (32) -> ... -> Pennsylvania Avenue (34) to Pennsylvania Avenue(31) -> Pacific Avenue(32) -> ... -> North Carolina Avenue (34)

Discussion:

#3: I read this as "hotels must be purchased before houses to improve a property".

#5: I read this as swapping the positions of these properties on the board without changing their values.

Assumptions:

- Player pays normal price for hotels.
- Hotels do not go away when a house is purchased.
- Rent for a property with hotel + houses is combined title rent.
- 2 hotels may not be purchased for a property.
- Due to component limitations (12 hotels in game), no more than 12 properties may be improved at one time.
- Maintaining even purchases across a color still applies, such that 3 hotels must be purchased on a 3-property color before the first house can be purchased.
- #4 replaces every space between Tennessee and Go (New York Avenue through Boardwalk, including chance, go to jail, and free parking)
- #4 and #5 conflict, so #5 must override #4 and the green properties are preserved (though out of order).
- The person who owns Tennessee avenue title collects rent on all of the Tennessee avenue spots.

- Tennessee Avenue spaces cannot be improved because New York Avenue can't be purchased and will never be auctioned off.

Assumed existing T-SAL action descriptions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
    (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
    (>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH))
    (NOT (SPACE-MORTGAGED ?SP))
  )
)

(:ACTION BUY_HOUSE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (EQ (MONOPOLY-MIN-HOUSES ?COLOR) ?MIN)
    (> (AVAILABLE-HOUSES) 0)
    (EQ (SPACE-HOUSES ?SP) ?MIN)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (+ ?MIN 1) ?NEW-HOUSES)
    (EQ (TITLE-HOUSES-RENT ?T ?NEW-HOUSES) ?NEW-RENT)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
```

```

    (SET (SPACE-HOUSES ?SP) ?NEW-HOUSES)
    (DECREASE (AVAILABLE-HOUSES) 1)
    (SET (SPACE-HOUSES ?SP) ?NEW-RENT)
  )
)

(:ACTION BUY_HOTEL
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4)
    (NOT (SPACE-HAS-HOTEL ?SP))
    (> (AVAILABLE-HOTELS) 0)
    (EQ (SPACE-HOUSES ?SP) 4)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) 0)
    (INCREASE (AVAILABLE-HOUSES) 4)
    (SPACE-HAS-HOTEL ?SP)
    (SET (SPACE-RENT ?SP) ?HOTEL-RENT)
  )
)

```

Estimated T-SAL description of novel actions

```

(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
  )
)

```

```

(EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
(>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
(>= (PLAYER-CASH ?P) (* 3 ?CASH))
)
:EFFECT
(AND
  (DECREASE (PLAYER-CASH ?P) (* 3 ?CASH))
  (NOT (SPACE-MORTGAGED ?SP))
)
)

(:ACTION BUY_HOUSE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (EQ (MONOPOLY-MIN-HOUSES ?COLOR) ?MIN)
    (> (AVAILABLE-HOUSES) 0)
    (EQ (SPACE-HOUSES ?SP) ?MIN)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (+ ?MIN 1) ?NEW-HOUSES)
    (EQ (TITLE-HOUSES-RENT ?T ?NEW-HOUSES) ?NEW-RENT)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
    (FORALL (?SAME-COLOR-SPACE - SPACE)
      (EQ (SPACE-COLOR ?SAME-COLOR-SPACE) ?COLOR)
      (SPACE-HAS-HOTEL ?SAME-COLOR-SPACE)
    )
  )
)
:EFFECT
(AND
  (DECREASE (PLAYER-CASH ?P) ?COST)
  (SET (SPACE-HOUSES ?SP) ?NEW-HOUSES)
  (DECREASE (AVAILABLE-HOUSES) 1)
  (SET (SPACE-HOUSES ?SP) (+ ?NEW-RENT ?HOTEL-RENT))
)
)

(:ACTION BUY_HOTEL

```

```
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4)
    (> (AVAILABLE-HOTELS) 0)
    (EQ (SPACE-HOUSES ?SP) 4)
    (EQ (SPACE-HOUSES ?SP) 0)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) 0)
    (INCREASE (AVAILABLE-HOUSES) 4)
    (SPACE-HAS-HOTEL ?SP)
    (SET (SPACE-RENT ?SP) ?HOTEL-RENT)
  )
)
```

Estimated T-SAL Transformation Sequence:

#1:

ADDRELATIONGENERATOR(PLAYER-CASH, ALLPERMUTATIONS([PLAYERS], CONSTANTFUNCTION(500))

#2:

REMOVEPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 1.1 ?CASH)))

ADDPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 3 ?CASH)))

REMOVEEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH)))

ADDEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 3 ?CASH)))

#3:

ADDPRECONDITION(BUY_HOUSE, (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT))

ADDPRECONDITION(BUY_HOUSE, (FORALL (?SAME-COLOR-SPACE - SPACE) (EQ (SPACE-COLOR ?SAME-COLOR-SPACE) ?COLOR) (SPACE-HAS-HOTEL ?SAME-COLOR-SPACE))))

REMOVEEFFECT(BUY_HOUSE, (SET (SPACE-HOUSES ?SP) ?NEW-RENT))

ADDEFFECT(BUY_HOUSE, (SET (SPACE-HOUSES ?SP) (+ ?NEW-RENT ?HOTEL-RENT)))

REMOVEPRECONDITION(BUY_HOTEL, (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4))

REMOVEPRECONDITION(BUY_HOTEL, (EQ (SPACE-HOUSES ?SP) 4))

ADDPRECONDITION(BUY_HOTEL, (EQ (SPACE-HOUSES ?SP) 0))

#4:

ADDVALUEGENERATOR(TENNSPACES, UNIFORMINTEGERDISTRIBUTION(20, 39))

ADDOBJECTGENERATOR(EXTRATENNS, OBJECTLIST(5, "PROP-4-2-"))

ADDERELATIONGENERATOR(POSITION-SPACE, ALLPERMUTATIONS([TENNSPACES], EXTRATENNS)

ADDERELATIONGENERATOR(SPACE-NUM, ALLPERMUTATIONS([EXTRATENNS], TENNSPACES)

ADDERELATIONGENERATOR(SPACE-TYPE, ALLPERMUTATIONS([EXTRATENNS], CONSTANTFUNCTION([PROPERTY])))

ADDERELATIONGENERATOR(SPACE-TITLE, ALLPERMUTATIONS([EXTRATENNS], TITLE-4-2)

#5:

ADDFLUENTVALUE(POSITION-SPACE, [31], PROP-7-3)

ADDFLUENTVALUE(POSITION-SPACE, [32], PROP-7-1)

ADDFLUENTVALUE(POSITION-SPACE, [34], PROP-7-2)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Missing Change, Added Changes:

Free_mortgage results in larger decrease in change value; free_mortgage doesn't work sometimes; buying a house doesn't work in states with monopolies and no hotels; buying a house does work in states with hotels; collection of rent happens in many unusual states to player with Tennessee Avenue

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level *does not match* intended level: This is not an environmental novelty according to our framework, as it does not change any environmental transitions (those outside of an agent's control). Examples of environmental transitions in Monopoly are draws from chance and community chest, die rolls, transitions to jail, rent payments, and auction starts.



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.

 *Simpler novelty **is possible***: This would be an environmental novelty by just modifying an effect of a chance or community chest card, so the complexity of composite events does not help meet environment novelty under our framework.

Note: Each numbered line can be assessed by itself as a novelty:

#1: object (Level 1)

#2: action (Level 3)

#3: interaction (Level 5) (because house is purchasable based on hotels on multiple properties)

#4: object (Level 1)

#5: object (Level 1)

Target Level: 6 - Environment

Novelty Description:

environments_2

1. The player will be charged 190% of the original property mortgage amount when the player want to free the mortgaged property.
2. Each player starts the game with 50 dollars
3. increase 50 more community cards "bank_error" and 50 more chance cards "general_repairs" on the board
4. rearrange the location sequence to: ['Virginia Avenue', 'Go', 'Community Chest', 'Mediterranean Avenue', 'Oriental Avenue', 'Income Tax', 'Reading Railroad', 'Baltic Avenue', 'Vermont Avenue', 'Connecticut Avenue', 'In Jail/Just Visiting', 'Chance', 'St. Charles Place', 'Electric Company', 'States Avenue', 'Pennsylvania Railroad', 'St. James Place', 'Community Chest', 'Tennessee Avenue', 'New York Avenue', 'Free Parking', 'Kentucky Avenue', 'Chance', 'Indiana Avenue', 'Illinois Avenue', 'B&O Railroad', 'Atlantic Avenue', 'Ventnor Avenue', 'Water Works', 'Marvin Gardens', 'Go to Jail', 'Pacific Avenue', 'North Carolina Avenue', 'Community Chest', 'Pennsylvania Avenue', 'Short Line', 'Chance', 'Park Place', 'Luxury Tax', 'Boardwalk']. All players start at location Virginia Avenue.
5. Change the mortgage amount of all real estate properties to 60 dollars

Assumptions:

Discussion:

#1: Every Community Chest and Chance card is a separate event, and there's no simple way of replacing them en masse, so this is a relatively complicated change.

Some changes are summarized below.

Assumed existing T-SAL action descriptions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P))
```

```
(EQ (SPACE-TITLE ?SP) ?TITLE)
(EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
(>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
)
:EFFECT
(AND
  (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH))
  (NOT (SPACE-MORTGAGED ?SP))
)
)
(:EVENT VISIT-CHANCE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TYPE ?SPACE) CHANCE)
    (ASSIGN ?CHANCE-CARD (:UNIFORM 1 17))
  )
:EFFECT
  (AND
    (VISIT-CHANCE ?P)
    (SET (CHANCE-CARD) ?CHANCE-CARD)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
(:EVENT CHEST-BANK-ERROR
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHEST-CARD) 4)
    (EQ (CURRENT-PLAYER) ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-CASH ?P) 200)
  )
)
(:EVENT VISIT-COMMUNITY-CHEST
:PARAMETERS ()
:PRECONDITION
```



```
(AND
  (EQ (CURRENT-PLAYER) ?P)
  (EQ (TURN-STAGE) VISIT)
  (EQ (PLAYER-POSITION ?P) ?POS)
  (EQ (POSITION-SPACE ?POS) ?SPACE)
  (EQ (SPACE-TYPE ?SPACE) CHEST)
  (ASSIGN ?CHEST-CARD (:UNIFORM 1 17))
)
:EFFECT
  (AND
    (VISIT-CHEST ?P)
    (SET (CHEST-CARD) ?CHEST-CARD)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
(:EVENT CHANCE-GENERAL-REPAIRS
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 8)
  )
:EFFECT
  (AND
    (NEED-MAINTENANCE 25 100)
  )
)
```

Estimated T-SAL description of novel actions

```
(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
    (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
(>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
    (>= (PLAYER-CASH ?P) (* 1.9 ?CASH))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 1.9 ?CASH))
```

```

        (NOT (SPACE-MORTGAGED ?SP))
    )
)
(:EVENT VISIT-CHANCE
:PARAMETERS ()
:PRECONDITION
    (AND
        (EQ (CURRENT-PLAYER) ?P)
        (EQ (TURN-STAGE) VISIT)
        (EQ (PLAYER-POSITION ?P) ?POS)
        (EQ (POSITION-SPACE ?POS) ?SPACE)
        (EQ (SPACE-TYPE ?SPACE) CHANCE)
        (ASSIGN ?CHANCE-CARD (:UNIFORM 1 67))
    )
:EFFECT
    (AND
        (VISIT-CHANCE ?P)
        (SET (CHANCE-CARD) ?CHANCE-CARD)
        (SET (TURN-STAGE) POST_ROLL)
    )
)
(:EVENT CHEST-BANK-ERROR-EXTRA
:PARAMETERS ()
:PRECONDITION
    (AND
        (> (CHEST-CARD) 17)
        (EQ (CURRENT-PLAYER) ?P)
    )
:EFFECT
    (AND
        (INCREASE (PLAYER-CASH ?P) 200)
    )
)
(:EVENT VISIT-COMMUNITY-CHEST
:PARAMETERS ()
:PRECONDITION
    (AND
        (EQ (CURRENT-PLAYER) ?P)
        (EQ (TURN-STAGE) VISIT)
        (EQ (PLAYER-POSITION ?P) ?POS)
        (EQ (POSITION-SPACE ?POS) ?SPACE)
        (EQ (SPACE-TYPE ?SPACE) CHEST)
        (ASSIGN ?CHEST-CARD (:UNIFORM 1 67))
    )

```

```

)
:EFFECT
  (AND
    (VISIT-CHEST ?P)
    (SET (CHEST-CARD) ?CHEST-CARD)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
(:EVENT CHANCE-GENERAL-REPAIRS-EXTRA
:PARAMETERS ()
:PRECONDITION
  (AND
    (> (CHANCE-CARD) 17)
  )
:EFFECT
  (AND
    (NEED-MAINTENANCE 25 100)
  )
)

```

Estimated T-SAL Transformation Sequence:

#1:

```

REMOVEPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 1.9 ?CASH)))
REMOVEEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 1.9 ?CASH)))

```

#2:

```

ADDRELATIONGENERATOR(PLAYER-CASH, ALLPERMUTATIONS([PLAYERS], CONSTANTFUNCTION(50)))

```

#3:

```

REMOVEPRECONDITION(VISIT-CHANCE, (ASSIGN ?CHANCE-CARD (:UNIFORM 1 17)))
ADDPRECONDITION(VISIT-CHANCE, (ASSIGN ?CHANCE-CARD (:UNIFORM 1 67)))
REMOVEPRECONDITION(VISIT-CHEST, (ASSIGN ?CHEST-CARD (:UNIFORM 1 17)))
ADDPRECONDITION(VISIT-CHEST, (ASSIGN ?CHEST-CARD (:UNIFORM 1 67)))
ADDEVENT(CHANCE-GENERAL-REPAIRS-EXTRA)
ADDPRECONDITION(CHANCE-GENERAL-REPAIRS-EXTRA, (> (CHANCE-CARD) 17))
ADDEFFECT(CHANCE-GENERAL-REPAIRS-EXTRA, (NEED-MAINTENANCE 25 100))
ADDEVENT(CHEST-BANK-ERROR-EXTRA)
ADDPRECONDITION(CHEST-BANK-ERROR-EXTRA, (> (CHEST-CARD) 17))
ADDEFFECT(CHEST-BANK-ERROR-EXTRA, (INCREASE (PLAYER-CASH ?P) 200))

```

#4:

```

ADDFLUENTVALUE(POSITION-SPACE, [0], PROP-3-3)
ADDFLUENTVALUE(POSITION-SPACE, [1], CORNER-GO)
ADDFLUENTVALUE(POSITION-SPACE, [2], CHEST-1)

```

...

#5:

ADDERELATIONGENERATOR(TITLE-MORTGAGE-VALUE, ALLPERMUTATIONS([TITLES], CONSTANTFUNCTION(60)))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Modified Change Rate, Missing Change, Added Changes: free_mortgage doesn't work sometimes; free_mortgage results in different decrease when it does work; visiting community chest and chance have changed rate of cash increase and maintenance demands, respectively; rolls result in different changes to rents and taxes; mortgaging gives different value.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.



Hierarchy Level Analysis

Analyzed level matches intended level: This is an environmental novelty, as it modifies environmental events (new chances of visiting cards).



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is possible: This would be an environmental novelty by just modifying an effect of a chance or community chest card, so the rest is unnecessary complexity.

Note: Each numbered line can be assessed by itself as a novelty:

#1: action (Level 3)

#2: object (Level 1)

#3: environmental (Level 6)

#4: object (Level 1) or relation (Level 4)

#5: object (Level 1) or action (Level 3)

Target Level: 6 - Environment

Novelty Description:

environments_3

2. Only one dice with number [1, 2] on the board. The number [1, 2] occurrence probability distribution is [1/2, 1/2].
3. Rearrange the sequence of Orchid color set from: St. Charles Place(11) -> ... -> States Avenue (13) -> Virginia Avenue (14) to Virginia Avenue(11) -> ... -> St. Charles Place(13) -> States Avenue (14)
4. There are 5 duplicated New York Avenue appearing after original New York Avenue on the board
5. Change the price of States Avenue from 140 to 50 dollars

Assumptions:

- #4 replaces every space between free parking and Illinois avenue, inclusive
- The person who owns New York avenue title collects rent on all of the New York Avenue spots.
- Every New York Avenue space is improved separately.

Discussion:

Assumed existing T-SAL action descriptions

```
(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (:UNIFORM 1 6))
  (SET (DIE2-VALUE) (:UNIFORM 1 6))
  (SET (TURN-STAGE) MOVE))
)
(:EVENT MOVE
:PARAMETERS ()
:PRECONDITION (AND (EQ (TURN-STAGE) MOVE) (EQ (CURRENT-PLAYER) ?P)
  (EQ (DIE1-VALUE) ?DIE1)
  (EQ (DIE2-VALUE) ?DIE2)
  (EQ (MAX-POSITION) ?MAX)
  (EQ (PLAYER-STATUS ?P) FREE))
:EFFECT (AND (SET (PLAYER-POSITION ?P) (MOD (+ ?DIE1 ?DIE2 (PLAYER-POSITION ?P)) ?MAX))
```

```
(SET (TURN-STAGE) VISIT))
)
```

Estimated T-SAL description of novel actions

```
(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (:UNIFORM 1 2))
  (SET (DIE2-VALUE) (:UNIFORM 1 6))
  (SET (DIE2-VALUE) 0)
  (SET (TURN-STAGE) MOVE))
)
```

Estimated T-SAL Transformation Sequence:

#2:

```
REMOVEEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (:UNIFORM 1 6)))
REMOVEEFFECT(DIE-ROLL, (SET (DIE2-VALUE) (:UNIFORM 1 6)))
ADDEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (:UNIFORM 1 2)))
ADDEFFECT(DIE-ROLL, (SET (DIE2-VALUE) 0))
```

#3:

```
ADDFLUENTVALUE(POSITION-SPACE, [11], PROP-3-3)
ADDFLUENTVALUE(POSITION-SPACE, [13], PROP-3-1)
ADDFLUENTVALUE(POSITION-SPACE, [14], PROP-3-2)
```

#4:

```
ADDVALUEGENERATOR(NYSPACES, UNIFORMINTEGERDISTRIBUTION(20, 24))
ADDOBJECTGENERATOR(EXTRANYS, OBJECTLIST(5, "PROP-4-3-"))
ADDRELATIONGENERATOR(POSITION-SPACE, ALLPERMUTATIONS([NYSPACES], EXTRANYS)
ADDRELATIONGENERATOR(SPACE-NUM, ALLPERMUTATIONS([EXTRANYS], NYSPACES)
ADDRELATIONGENERATOR(SPACE-TYPE, ALLPERMUTATIONS([EXTRANYS], CONSTANTFUNCTION([PROPERTY]))
ADDRELATIONGENERATOR(SPACE-TITLE, ALLPERMUTATIONS([EXTRANYS], TITLE-4-3)
```

#5:

```
ADDFLUENTVALUE(TITLE-COST, [TITLE-3-2], 50)
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Modified Change Rate: different die rolls; rolls result in different rents; purchasing states avenue gives different decrease in cash.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is an environmental novelty, as it modifies environmental events (die rolls).
- ✓ *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
- ✗ *Simpler novelty **is possible**:* This would be an environmental novelty by just modifying an effect of a chance or community chest card, so the rest is unnecessary complexity.

Note: Each numbered line can be assessed by itself as a novelty:

#2: environmental (Level 6)

#3: object (Level 1)

#4: relation (Level 4) (modifications to static space-title relationship)

#5: object (Level 1)

Target Level: 6 - Environment

Novelty Description:

environments_4

1. Applying card effect on all the players when one player pick up a community or chance card.
2. There are 20 duplicated Tennessee Avenue appearing after original Tennessee Avenue on the board.
3. Rearrange the sequence of Green color set from: Pacific Avenue(31) -> North Carolina Avenue (32) -> ... -> Pennsylvania Avenue (34) to Pennsylvania Avenue(31) -> Pacific Avenue(32) -> ... -> North Carolina Avenue (34)
4. Both dices have odd number only. The number [1, 3, 5] occurrence probability distribution of both dices are [1/3, 1/3, 1/3].
5. The player will be charged 300% of the total debt of all mortgaged properties when the player want to free one mortgaged property.

Assumptions:

- #1 The “pick-up” effect of a get out of jail free card is getting the card to use later, not getting you out of jail
- #2 replaces every space between Tennessee and Go (New York Avenue through Boardwalk, including chance, go to jail, and free parking)
- #5 is all properties, not all of the player’s properties
- #2 and #3 conflict, so #3 must override #2 and the green properties are preserved (though out of order).
- The person who owns Tennessee avenue title collects rent on all of the Tennessee avenue spots.
- Tennessee Avenue spaces cannot be improved because New York Avenue can’t be purchased and will never be auctioned off.

Discussion:

#1: Every Community Chest and Chance card is a separate event, and there’s no simple way of replacing them en masse, so this is a relatively complicated change.

Some changes are summarized below.

Assumed existing T-SAL action descriptions

```
(:EVENT CHANCE-GO-TO-JAIL
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 1)
```

```
(VISITING-CHANCE ?P)
)
:EFFECT
  (AND
    (SET (PLAYER-STATUS ?P) CAUGHT)
  )
)

(:EVENT CHANCE-GET-OUT-OF-JAIL-FREE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 2)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-GET-OUT-OF-JAIL-FREES ?P) 1)
  )
)

(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (:UNIFORM 1 6))
  (SET (DIE2-VALUE) (:UNIFORM 1 6))
  (SET (TURN-STAGE) MOVE))
)

(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
    (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
    (>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH))
```

```
(NOT (SPACE-MORTGAGED ?SP))
)
)
```

Estimated T-SAL description of novel actions

```
(:EVENT CHANCE-GO-TO-JAIL
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 1)
    (VISITING-CHANCE ?ANYONE)
    (IS-PLAYER ?P)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (SET (PLAYER-STATUS ?P) CAUGHT)
  )
)

(:EVENT CHANCE-GET-OUT-OF-JAIL-FREE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 2)
    (VISITING-CHANCE ?ANYONE)
    (IS-PLAYER ?P)
    (VISITING-CHANCE ?P)
  )
:EFFECT
  (AND
    (INCREASE (PLAYER-GET-OUT-OF-JAIL-FREES ?P) 1)
  )
)

(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (- (* 2 (:UNIFORM 1 3) 1))
  (SET (DIE2-VALUE) (- (* 2 (:UNIFORM 1 3) 1))
  (SET (TURN-STAGE) MOVE))
)

(:ACTION FREE_MORTGAGE
:PARAMETERS (?SP - SPACE)
```

```
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (SPACE-MORTGAGED ?SP)
    (EQ (SPACE-OWNED ?SP) ?P)
    (EQ (SPACE-TITLE ?SP) ?TITLE)
(EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH)
(>= (PLAYER-CASH ?P) (* 1.1 ?CASH))
    (SUM ?TOTAL (?MTG-SP - SPACE)
      (AND (SPACE-MORTGAGED ?MTG-SP) (EQ (SPACE-TITLE ?MTG-SP) ?MTG-TITLE)
        (TITLE-MORTGAGE-VALUE ?MTG-TITLE)
      )
    )
    (>= (PLAYER-CASH ?P) (* 3 ?TOTAL))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* 3 ?TOTAL))
    (NOT (SPACE-MORTGAGED ?SP))
  )
)
```

Estimated T-SAL Transformation Sequence:

#1:

```
ADDPRECONDITION(CHANCE-GO-TO-JAIL, (VISITING-CHANCE ?ANYONE))
ADDPRECONDITION(CHANCE-GO-TO-JAIL, (IS-PLAYER ?P))
REMOVEPRECONDITION(CHANCE-GO-TO-JAIL, (VISITING-CHANCE ?P))
ADDPRECONDITION(CHANCE-GET-OUT-OF-JAIL-FREE, (VISITING-CHANCE ?ANYONE))
ADDPRECONDITION(CHANCE-GET-OUT-OF-JAIL-FREE, (IS-PLAYER ?P))
REMOVEPRECONDITION(CHANCE-GET-OUT-OF-JAIL-FREE, (VISITING-CHANCE ?P))
... (All other chance and chest cards changed similarly)
```

#2:

```
ADDVALUEGENERATOR(TENNSPACES, UNIFORMINTEGERDISTRIBUTION(20, 39))
ADDOBJECTGENERATOR(EXTRATENNS, OBJECTLIST(5, "PROP-4-2-"))
ADDRELATIONGENERATOR(POSITION-SPACE, ALLPERMUTATIONS([TENNSPACES], EXTRATENNS)
ADDRELATIONGENERATOR(SPACE-NUM, ALLPERMUTATIONS([EXTRATENNS], TENNSpaces)
ADDRELATIONGENERATOR(SPACE-TYPE, ALLPERMUTATIONS([EXTRATENNS], CONSTANTFUNCTION([PROPERTY]))
ADDRELATIONGENERATOR(SPACE-TITLE, ALLPERMUTATIONS([EXTRATENNS], TITLE-4-2)
```

#3:

```
ADDFLUENTVALUE(POSITION-SPACE, [31], PROP-7-3)
ADDFLUENTVALUE(POSITION-SPACE, [32], PROP-7-1)
ADDFLUENTVALUE(POSITION-SPACE, [34], PROP-7-2)
```

#4:

```
REMOVEEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (:UNIFORM 1 6)))
REMOVEEFFECT(DIE-ROLL, (SET (DIE2-VALUE) (:UNIFORM 1 6)))
ADDEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (- (* 2 (:UNIFORM 1 3)) 1)))
ADDEFFECT(DIE-ROLL, (SET (DIE2-VALUE) (- (* 2 (:UNIFORM 1 3)) 1)))
#5:
REMOVEPRECONDITION(FREE_MORTGAGE, (EQ (TITLE-MORTGAGE-VALUE ?TITLE) ?CASH))
REMOVEPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDPRECONDITION(FREE_MORTGAGE, (SUM ?TOTAL (?MTG-SP - SPACE) (AND (SPACE-MORTGAGED ?MTG-SP)
    (EQ (SPACE-TITLE ?MTG-SP) ?MTG-TITLE) (TITLE-MORTGAGE-VALUE ?MTG-TITLE))))
ADDPRECONDITION(FREE_MORTGAGE, (>= (PLAYER-CASH ?P) (* 3 ?TOTAL)))
REMOVEEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 1.1 ?CASH)))
ADDEFFECT(FREE_MORTGAGE, (DECREASE (PLAYER-CASH ?P) (* 3 ?TOTAL)))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Change in Probability, Missing Change, Added Changes: Some dice values are no longer possible; free_mortgage results in larger decrease in change value; free_mortgage doesn't work sometimes; values change when other players go to chance or community chest

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: Eventually an agent will draw get out of jail free, and a new state will occur where everyone has a get out of jail free card.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.



Hierarchy Level Analysis

Analyzed level matches intended level: This is an environmental novelty, as it modifies environmental events (chance, community chest, and die rolls).



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is possible: This would be an environmental novelty by just modifying

an effect of a chance or community chest card, so the rest is unnecessary complexity.

Note: Each numbered line can be assessed by itself as a novelty:

#1: environmental (Level 6)

#2: relation (level 4)

#3: object (Level 1) or relation (level 4)

#4: environmental (Level 6)

#5: action (Level 3)

Target Level: 6 - Environment

Novelty Description:

environments_5

1. Increase 50 more community card "birthday" and 50 more chance card "building_loan_matures" on the board
2. Change the price of the house of Boardwalk from 200 to 10 dollars
3. Change the available houses in bank from 32 to 9
5. Only one dice with number [1, 2, 5, 6] on the board. The number [1, 2, 5, 6] occurrence probability distribution is [1/4, 1/4, 1/4, 1/4].

Assumptions:

- # Available houses is public information, so is price of house on Boardwalk

Discussion:

Estimated T-SAL description of modified event

```
(:EVENT DIE-ROLL
:PARAMETERS ()
:PRECONDITION (EQ (TURN-STAGE) ROLL)
:EFFECT (AND
  (SET (DIE1-VALUE) (:DRAWFROMSET (1 2 5 6)))
  (SET (DIE2-VALUE) (:DRAWFROMSET (1 2 5 6)))
  (SET (TURN-STAGE) MOVE))
)
```

Estimated T-SAL Transformation Sequence:

#1:

```
REMOVEPRECONDITION(VISIT-CHANCE, (ASSIGN ?CHANCE-CARD (:UNIFORM 1 17)))
ADDPRECONDITION(VISIT-CHANCE, (ASSIGN ?CHANCE-CARD (:UNIFORM 1 67)))
REMOVEPRECONDITION(VISIT-CHEST, (ASSIGN ?CHEST-CARD (:UNIFORM 1 17)))
ADDPRECONDITION(VISIT-CHEST, (ASSIGN ?CHEST-CARD (:UNIFORM 1 67)))
ADDEVENT(CHANCE-BUILDING-LOAN-MATURES-EXTRA)
ADDPRECONDITION(CHANCE-BUILDING-LOAN-MATURES-EXTRA, (> (CHANCE-CARD) 17))
ADDEFFECT(CHANCE-BUILDING-LOAN-MATURES-EXTRA, (INCREASE (PLAYER-CASH ?P) 150))
ADDEVENT(CHEST-BIRTHDAY-EXTRA)
ADDPRECONDITION(CHEST-BIRTHDAY-EXTRA, (> (CHEST-CARD) 17))
ADDEFFECT(CHEST-BIRTHDAY-EXTRA, (NEED-COLLECTION 10))
```

#2:

```
ADDFLUENTVALUE(TITLE-HOUSE-COST, [TITLE-8-2], 10)
```

#3:

ADDERELATIONGENERATOR(AVAILABLE-HOUSES, ALLPERMUTATIONS([], CONSTANTFUNCTION(9))

#5:

REMOVEEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (:UNIFORM 1 6)))

REMOVEEFFECT(DIE-ROLL, (SET (DIE2-VALUE) (:UNIFORM 1 6)))

ADDEFFECT(DIE-ROLL, (SET (DIE1-VALUE) (DRAWFROMSET (1 2 5 6))))

ADDEFFECT(DIE-ROLL, (SET (DIE2-VALUE) (DRAWFROMSET (1 2 5 6))))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change value, Missing Change, Modified change rate: Buying house on Boardwalk decreases funds more; buying a house fails earlier; die rolls have different distribution

Novelty Entry Determinism: None Novelty exists immediately since the # available houses changes.

Novelty Obviousness: Alien Observation # available houses changes observation immediately.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to # available houses.

Novel State Observation Activity: Passive: All states are new, but immediately observable.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This is an environmental novelty, as it modifies environmental events (new chances of visiting cards).



Higher level novelties are not present: Agent goals do not change. No modifications to non-environmental events.



Simpler novelty is possible: This would be an environmental novelty by just modifying an effect of a chance or community chest card, so the rest is unnecessary complexity.

Note: Each numbered line can be assessed by itself as a novelty:

#1: environmental (Level 6)

#2: object (Level 1)

#3: action (Level 3) or none

#5: environmental (Level 6)

Target Level: 6 - Environment

Novelty Description:

environments_12

1. Each player has to improve hotel before improving house
2. Rearrange the sequence of Yellow color set from: Atlantic Avenue(26) -> ... -> Ventnor Avenue (28) -> Marvin Avenue (29) to Marvin Avenue(26) -> ... -> Atlantic Avenue(28) -> Ventnor Avenue (29)
3. The player will receive 100 dollars instead of paying 50 dollars when pick up community card "doctor_fee". The player will receive 100 dollars instead of receiving 50 dollars when pick up chance card "bank_dividend".

Discussion:

Interpretation:

1. I read this as "hotels must be purchased before houses to improve a property".
2. I read this as swapping the positions of these properties on the board without changing their values.
3. This likely meant "pay 100 dollars" for "doctor fee", encoding it as such.

Assumptions:

- Player pays normal price for hotels.
- Hotels do not go away when a house is purchased.
- Rent for a property with hotel + houses is combined title rent.
- 2 hotels may not be purchased for a property.
- Due to component limitations (12 hotels in game), no more than 12 properties may be improved at one time.
- Maintaining even purchases across a color still applies, such that 3 hotels must be purchased on a 3-property color before the first house can be purchased.

Assumed existing T-SAL action descriptions

```
(:ACTION BUY_HOUSE
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (EQ (MONOPOLY-MIN-HOUSES ?COLOR) ?MIN)
    (> (AVAILABLE-HOUSES) 0)
    (EQ (SPACE-HOUSES ?SP) ?MIN)
    (EQ (SPACE-TITLE ?SP) ?T))
```

```

    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (+ ?MIN 1) ?NEW-HOUSES)
    (EQ (TITLE-HOUSES-RENT ?T ?NEW-HOUSES) ?NEW-RENT)
  )
  :EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) ?NEW-HOUSES)
    (DECREASE (AVAILABLE-HOUSES) 1)
    (SET (SPACE-HOUSES ?SP) ?NEW-RENT)
  )
)

(:ACTION BUY_HOTEL
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4)
    (NOT (SPACE-HAS-HOTEL ?SP))
    (> (AVAILABLE-HOTELS) 0)
    (EQ (SPACE-HOUSES ?SP) 4)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
  )
  :EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) 0)
    (INCREASE (AVAILABLE-HOUSES) 4)
    (SPACE-HAS-HOTEL ?SP)
    (SET (SPACE-RENT ?SP) ?HOTEL-RENT)
  )
)

(:EVENT CHANCE-BANK-DIVIDEND
:PARAMETERS ()
:PRECONDITION
  (AND

```

```

      (EQ (CHANCE-CARD) 15)
      (VISITING-CHANCE ?P)
    )
  :EFFECT
    (AND
      (INCREASE (PLAYER-CASH ?P) 50)
    )
  )
  (:EVENT CHEST-DOCTOR-FEE
  :PARAMETERS ()
  :PRECONDITION
    (AND
      (EQ (CHEST-CARD) 5)
      (VISITING-CHEST ?P)
    )
  :EFFECT
    (AND
      (DECREASE (PLAYER-CASH ?P) 50)
    )
  )
)

```

Estimated T-SAL description of novel actions

```

  (:ACTION BUY_HOUSE
  :PARAMETERS (?SP - SPACE)
  :PERFORMER ?P
  :PRECONDITION
    (AND
      (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
      (EQ (SPACE-COLOR ?SP) ?COLOR)
      (EQ (MONOPOLY-OWNED ?COLOR) ?P)
      (EQ (MONOPOLY-MIN-HOUSES ?COLOR) ?MIN)
      (> (AVAILABLE-HOUSES) 0)
      (EQ (SPACE-HOUSES ?SP) ?MIN)
      (EQ (SPACE-TITLE ?SP) ?T)
      (EQ (TITLE-HOUSE-COST ?T) ?COST)
      (>= (PLAYER-CASH ?P) ?COST)
      (EQ (+ ?MIN 1) ?NEW-HOUSES)
      (EQ (TITLE-HOUSES-RENT ?T ?NEW-HOUSES) ?NEW-RENT)
      (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT)
      (FORALL (?SAME-COLOR-SPACE - SPACE)
        (EQ (SPACE-COLOR ?SAME-COLOR-SPACE) ?COLOR)
        (SPACE-HAS-HOTEL ?SAME-COLOR-SPACE)
      )
    )
  )
  :EFFECT

```

```

(AND
  (DECREASE (PLAYER-CASH ?P) ?COST)
  (SET (SPACE-HOUSES ?SP) ?NEW-HOUSES)
  (DECREASE (AVAILABLE-HOUSES) 1)
  (SET (SPACE-HOUSES ?SP) (+ ?NEW-RENT ?HOTEL-RENT)))
)
)

(:ACTION BUY_HOTEL
:PARAMETERS (?SP - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (OR (EQ (TURN-STAGE) PRE_ROLL) (EQ (TURN-STAGE) POST_ROLL))
    (EQ (SPACE-COLOR ?SP) ?COLOR)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
(>= (MONOPOLY-MIN-HOUSES ?COLOR) 4)
    (> (AVAILABLE-HOTELS) 0)
(EQ (SPACE-HOUSES ?SP) 4)
    (EQ (SPACE-HOUSES ?SP) 0)
    (EQ (SPACE-TITLE ?SP) ?T)
    (EQ (TITLE-HOUSE-COST ?T) ?COST)
    (>= (PLAYER-CASH ?P) ?COST)
    (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-HOUSES ?SP) 0)
    (INCREASE (AVAILABLE-HOUSES) 4)
    (SPACE-HAS-HOTEL ?SP)
    (SET (SPACE-RENT ?SP) ?HOTEL-RENT))
  )
)

(:EVENT CHANCE-BANK-DIVIDEND
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHANCE-CARD) 15)
    (VISITING-CHANCE ?P))
  )
:EFFECT
  (AND

```

```

    (INCREASE (PLAYER-CASH ?P) 100)
  )
)
(:EVENT CHEST-DOCTOR-FEE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CHEST-CARD) 5)
    (VISITING-CHEST ?P)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) 100)
  )
)

```

Estimated T-SAL Transformation Sequence:

#1:

```

ADDPRECONDITION(BUY_HOUSE, (EQ (TITLE-HOTEL-RENT ?T) ?HOTEL-RENT))
ADDPRECONDITION(BUY_HOUSE, (FORALL (?SAME-COLOR-SPACE - SPACE) (EQ (SPACE-COLOR ?SAME-COLOR-SPACE) ?COLOR) (SPACE-HAS-HOTEL ?SAME-COLOR-SPACE))))
REMOVEEFFECT(BUY_HOUSE, (SET (SPACE-HOUSES ?SP) ?NEW-RENT))
ADDEFFECT(BUY_HOUSE, (SET (SPACE-HOUSES ?SP) (+ ?NEW-RENT ?HOTEL-RENT)))
REMOVEPRECONDITION(BUY_HOTEL, (>= (MONOPOLY-MIN-HOUSES ?COLOR) 4))
REMOVEPRECONDITION(BUY_HOTEL, (EQ (SPACE-HOUSES ?SP) 4))
ADDPRECONDITION(BUY_HOTEL, (EQ (SPACE-HOUSES ?SP) 0))

```

#2:

```

ADDFLUENTVALUE(POSITION-SPACE, [26], PROP-6-3)
ADDFLUENTVALUE(POSITION-SPACE, [28], PROP-6-1)
ADDFLUENTVALUE(POSITION-SPACE, [29], PROP-6-2)

```

#3:

```

REMOVEEFFECT(CHANCE-BANK-DIVIDEND, (INCREASE (PLAYER-CASH ?P) 50))
ADDEFFECT(CHANCE-BANK-DIVIDEND, (INCREASE (PLAYER-CASH ?P) 100))
REMOVEEFFECT(CHEST-DOCTOR-FEE, (DECREASE (PLAYER-CASH ?P) 50))
ADDEFFECT(CHEST-DOCTOR-FEE, (DECREASE (PLAYER-CASH ?P) 100))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Value Some community chest and chance changes are increased in size; purchasing houses has an increase in the rent value; paying rent on Atlantic Avenue / Marvin Gardens increases and decreases, respectively

Novelty Entry Determinism: None Novelty exists immediately since the yellow properties are switched.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to yellow properties.

Novel State Observation Activity: Active: Agent must attempt to buy a hotel to observe a hotel on one property of monopoly, nothing on others.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is an environmental novelty, as it modifies the effects of environmental events (chest-doctor-fee and chance-bank-dividend).
- ✓ *Higher level novelties are not present:* Agent goals do not change. No modifications to non-environmental events.
- ✗ *Simpler novelty **is possible**:* This would be an environmental novelty by just modifying an effect of one card, so the rest is unnecessary complexity.

Note: In our framework, the “buy hotel” and “buy house” changes alone are action novelty, and the fluent value changes by themselves are object novelties.

Monopoly Individual Level 7 Novelties

We treat the three different agents described for the goal level here, corresponding to the goals_1, goals_2, and goals_3 rows.

Target Level: 7 - goals

Novelty Description:

Goals_1

The three agents play with TA2 will be background_agent_v3_1, cash_conservator, and background_agent_v4_1

This agent is cash conservative, so it will try its best to conserve its cash.

Specific actions:

1. auction: never attend in auction
2. trade: never make or accept trade with others
3. free_mortgage: never free mortgage
4. pay_jail_fine: never pay jail fine, rather stay in jail
5. buy_property: it will only buy property if the price $\leq$ 10% cash
6. improve_property: it will improve property only if the price $\leq$ 10% cash

Discussion:

We'll approximate the standard monopoly goal by saying that the agent gets a positive reward for having positive cash and being the only agent with cash. In addition, the cash conservative player receives a massive negative reward every time it bids in an auction, makes or accepts a trade offer, frees a mortgage, pays a jail fine, purchases a property if the price is more than 10% of the player's cash, or buys a house or hotel for more than 10% of cash on hand.

Note that below, the function "F" is the linear temporal logic symbol for "eventually this state holds", and "chooses" represents the agent selecting an action in a state. For the following logical statement:

$$(F (AND (CHOOSSES ?AG (PURCHASE ?SP)) (EQ (SPACE-TITLE ?SP) ?T) (> (TITLE-COST ?T) (* .1 (PLAYER-CASH ?AG)))))$$

a correct interpretation is "at any time during the game agent ?AG chooses to purchase a property ?SP with a title ?T whose cost is greater than 0.1 times ?AG's total cash available in the state the choice was made".

Assumptions:

Estimated T-SAL Transformation Sequence:


```

REPLACEPERFORMANCEEQUATION(CASH_CONSERVATOR,
    '(+ (IF (F (AND (> (PLAYER-CASH ?AG) 0)
        (>= (PLAYER-CASH ?AG)
            (SUM ?OAG (IS-PLAYER ?OAG) (PLAYER-CASH ?OAG))))))
    1 OTHERWISE 0
    (IF (F (CHOOSES ?AG (BID-IN-AUCTION ?AMT))) -100000 OTHERWISE 0)
    (IF (F (CHOOSES ?AG (MAKE_TRADE_OFFER ?OFFER))) -100000 OTHERWISE 0)
    (IF (F (CHOOSES ?AG (ACCEPT_TRADE_OFFER ?OFFER))) -100000 OTHERWISE 0)
    (IF (F (CHOOSES ?AG (FREE_MORTGAGE ?SP))) -100000 OTHERWISE 0)
    (IF (F (CHOOSES ?AG (PAY_JAIL_FINE))) -100000 OTHERWISE 0)
    (IF (F (AND (CHOOSES ?AG (PURCHASE ?SP)) (EQ (SPACE-TITLE ?SP) ?T)
        (> (TITLE-COST ?T) (* .1 (PLAYER-CASH ?AG)))))
        -100000 OTHERWISE 0)
    (IF (F (AND (CHOOSES ?AG (BUY_HOUSE ?SP)) (EQ (SPACE-TITLE ?SP) ?T)
        (> (TITLE-HOUSE-COST ?T) (* .1 (PLAYER-CASH ?AG)))))
        -100000 OTHERWISE 0)
    (IF (F (AND (CHOOSES ?AG (BUY_HOTEL ?SP)) (EQ (SPACE-TITLE ?SP) ?T)
        (> (TITLE-HOUSE-COST ?T) (* .1 (PLAYER-CASH ?AG)))))
        -100000 OTHERWISE 0)

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Uncertain This agent may make actions out of distribution of standard agents, don't know yet.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is a goal novelty, as only goals change.
- ✓ *Higher level novelties are not present:* No modifications to events (or physics at all).
- ✓ *Simpler novelty is not possible:* Our framework represents this as a change in a performance equation to a different performance equation, so no simpler change is possible.

Target Level: 7 - goals

Novelty Description:

Goals_2

The three agents play with TA2 will be background_agent_v3_1, railroad_chaser, and background_agent_v4_1

This agent want to own railroad as its best.

Specific actions:

1. auction: make more bid if asset is railroad
2. buy property: if asset is railroad and still have positive cash balance afterwards, then buy it.
3. trade: never offer owned railroad to others, only make offer for railroad from others.

Discussion:

We'll approximate this by saying that the agent has higher performance for every railroad it owns, and it specifically gets a massive negative reward for failing to purchase a railroad or making or accepting a trade offer in which it loses a railroad.

Assumptions:

Estimated T-SAL Transformation Sequence:

```
REPLACEPERFORMANCEEQUATION(RAILROAD_CHASER,
  '(+ (IF (F (AND (> (PLAYER-CASH ?AG) 0)
    (>= (PLAYER-CASH ?AG)
      (SUM ?OAG (IS-PLAYER ?OAG) (PLAYER-CASH ?OAG))))))
    100 OTHERWISE 0)
  (SUM ?RR (AND (EQ (SPACE-OWNED ?RR) ?AG) (EQ (SPACE-TYPE ?RR) RAILROAD)) 10)
  (IF (F (AND (CHOOSES ?AG (MAKE_TRADE_OFFER ?OFFER))
    (OFFER-INCLUDES-SPACE ?SP)
    (EQ (SPACE-TYPE ?SP) RAILROAD)))
    -100000 OTHERWISE 0)
  (IF (F (AND (CHOOSES ?AG (ACCEPT_TRADE_OFFER ?OFFER))
    (OFFER-REQUESTS-SPACE ?SP)
    (EQ (SPACE-TYPE ?SP) RAILROAD)))
    -100000 OTHERWISE 0)
  (IF (F (AND (NOT (CHOOSES ?AG (PURCHASE ?SP)))
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?AG) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SP)
    (EQ (PLAYER-OWNED ?SP) NO-PLAYER)
    (EQ (SPACE-TYPE ?SP) RAILROAD))
```

(EQ (SPACE-TITLE ?SP) ?TITLE)
(< (TITLE-COST ?T) (PLAYER-CASH ?AG))))
-100000 OTHERWISE 0)))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Uncertain This agent may make actions out of distribution of standard agents, don't know yet.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is a goal novelty, as only goals change.
- ✓ *Higher level novelties are not present:* No modifications to events (or physics at all).
- ✓ *Simpler novelty is not possible:* Our framework represents this as a change in a performance equation to a different performance equation, so no simpler change is possible.

Target Level: 7 - goals

Novelty Description:

The three agents play with TA2 will be background_agent_v3_1, sneaky_trader, and background_agent_v4_1

This agent is sneaky trader that offer small amount money (\$ 0.1) to trade other's property.

Discussion:

We'll approximate this by saying that the agent has higher performance for every time a \$0.1 offer made by itself is accepted by another player.

Estimated T-SAL original action:

```
(:ACTION ACCEPT_TRADE_OFFER
:PARAMETERS (?OFFER - OFFER)
:PERFORMER ?P
:PRECONDITION
  (AND
    (EQ (OFFER-PARTNER ?OFFER) ?P)
    (NOT (OFFER-CANCELLED ?OFFER))
    (OFFER-STILL-WORKS ?OFFER)
  )
:EFFECT (AND (OFFER-PROCESSING ?OFFER) (OFFER-ACCEPTED ?OFFER))
)
```

Estimated T-SAL Transformation Sequence:

```
REPLACEPERFORMANCEEQUATION(SNEAKY_TRADER,
  '(+ (IF (F (AND (> (PLAYER-CASH ?AG) 0)
    (>= (PLAYER-CASH ?AG)
      (SUM ?OAG (IS-PLAYER ?OAG) (PLAYER-CASH ?OAG))))))
    100 OTHERWISE 0
  (SUM ?OFFER (AND (OFFER-ACCEPTED ?OFFER) (EQ (OFFER-OFFERER ?OFFER) ?AGENT))
    (EQ (OFFER-INCLUDES-CASH ?OFFER) 0.1)
    (NOT (OFFER-INCLUDES-PROPERTY ?OFFER ?ANY))
    (OFFER-REQUESTS-PROPERTY ?OFFER ?SOME))
  1)))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Uncertain This agent may make actions out of distribution of standard agents, don't know yet.

Novelty Entry Determinism: None Novelty exists immediately since the properties are changed.

Novelty Obviousness: Alien Observation Switched properties create out-of-original-distribution observations.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Distant Agent observes novelty before acting, due to changed properties.

Novel State Observation Activity: Passive: All states are distinguishable immediately.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is a goal novelty, as only goals change.
- ✓ *Higher level novelties are not present:* No modifications to events (or physics at all).
- ✓ *Simpler novelty is not possible:* Our framework represents this as a change in a performance equation to a different performance equation, so no simpler change is possible.

Monopoly Combined Level 7 Novelties

The rest of the rows are combinations of the above individual novelties, and our analyses of them do not differ. Here's our analysis of the combinations in general.

Target Level: 7 - goals

Novelty Description:

goals_4-15

All remaining novelties are combinations of the agents described previously.

Hierarchy Level Analysis



Analyzed level matches intended level: This is a goal novelty, as only goals change.



Higher level novelties are not present: No modifications to events (or physics at all).



*Simpler novelty **is possible**:* Our framework represents this as a change in multiple performance equations, and any change by itself is a goal novelty.

Monopoly Individual Level 8 Novelties

Below we analyze each of the individual novelties that make up the combined novelties you sent.

Target Level: 8 - Events

Novelty Description:

avenue_rule_change, min_cash_limit = 100

A player can not buy the Avenue location if not having cash with 100 dollars

Discussion:

Interpretation:

No property can be purchased without a minimum amount of cash available.

Different amounts of minimum cash limit are chosen for different experiments; the constant value 100 is used below, and can be replaced without meaningful detriment to the analysis.

Assumed existing T-SAL action descriptions

```
(:ACTION PURCHASE
:PARAMETERS (?SPACE - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (EQ (TURN-STAGE) VISIT)
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TITLE ?SPACE) ?TITLE)
    (NEQ (SPACE-TITLE ?SPACE) NOTITLE)
    (EQ (SPACE-OWNED ?SPACE) NOPLAYER)
    (< (TITLE-COST ?TITLE) (PLAYER-CASH ?P))
    (EQ (TITLE-RENT-UNIMPROVED ?TITLE) ?NEW-RENT)
    (EQ (TITLE-COST ?TITLE) ?COST)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-OWNED ?SPACE) ?P)
    (SET (SPACE-RENT ?SPACE) ?NEW-RENT)
```



```

    (SET (TURN-STAGE) POST_ROLL)
  )
)

```

Estimated T-SAL description of novel actions

```

(:ACTION PURCHASE
:PARAMETERS (?SPACE - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (EQ (TURN-STAGE) VISIT)
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TITLE ?SPACE) ?TITLE)
    (NEQ (SPACE-TITLE ?SPACE) NOTITLE)
    (EQ (SPACE-OWNED ?SPACE) NOPLAYER)
    (< (TITLE-COST ?TITLE) (PLAYER-CASH ?P))
    (> (PLAYER-CASH ?P) 100)
    (EQ (TITLE-RENT-UNIMPROVED ?TITLE) ?NEW-RENT)
    (EQ (TITLE-COST ?TITLE) ?COST)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-OWNED ?SPACE) ?P)
    (SET (SPACE-RENT ?SPACE) ?NEW-RENT)
    (SET (TURN-STAGE) POST_ROLL)
  )
)

```

Estimated T-SAL Transformation Sequence:

ADDPRECONDITION(PURCHASE, (> (PLAYER-CASH ?P) 100))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change Sometimes the purchase action simply fails, and no changes occur.

Novelty Entry Determinism: Infrequent Nondeterministic: The proximate cause of entering a state with a purchase decision is a nondeterministic roll. The agent's chance of landing on an unowned property generally decreases over the course of the game. However, for minimum cash values greater than the agent's starting money, this will be **Frequent Nondeterministic**, at least for initial entries.

Novelty Obviousness: Novel Transition Discernible Transitions in which property purchases fail should be immediately obvious.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until it attempts and fails to make a purchase.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

 **Analyzed level *does not match* intended level:** We model this most easily as an action novelty, because an action precondition is directly affected.

Possible remedy: This would seem more like an event novelty if something out of the agent's control happened immediately after the purchase instead of the purchase being prevented; for example, the property just purchased might be mortgaged immediately.

 **Higher level novelties are not present:** No higher levels exist.

 **Simpler novelty is not possible:** This novelty primitive corresponds to a single transformation, and cannot be reduced.

Target Level: 8 - Events

Novelty Description:

alternate_income_tax_payment

If a player is landing on location 'Income tax' and also owns at least one utility, then the tax is exempted.

Discussion:

Normally, rent is charged on the income tax spot because rent is > 0; in order to charge differently, we need to remove rent from this spot and create a new event.

Assumed existing T-SAL action descriptions

```
(:EVENT PAY
:PARAMETERS ()
:PRECONDITION (AND (EQ (CURRENT-PLAYER) ?P)
  (EQ (TURN-STAGE) VISIT)
  (EQ (PLAYER-POSITION ?P) ?POS)
  (EQ (POSITION-SPACE ?POS) ?SPACE)
  (EQ (SPACE-OWNED ?SPACE) ?OWNER)
  (EQ (SPACE-RENT ?SPACE) ?COST)
  (> (SPACE-RENT ?SPACE) 0)
)
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (INCREASE (PLAYER-CASH ?OWNER) ?COST)
  )
)
```

Estimated T-SAL description of novel actions

```
(:EVENT PAY-INCOME-TAX
:PARAMETERS ()
:PRECONDITION
  (AND (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) INCOME-TAX)
    (FORALL (?SP — SPACE) (EQ (SPACE-TYPE ?SP) UTILITY) (NOT (EQ (SPACE-OWNER ?SP) ?P))))
)
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) 200)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDFLUENTVALUE(SPACE-RENT, [INCOME-TAX], 0)
ADDEVENT(PAY-INCOME-TAX)
ADDPRECONDITION(PAY-INCOME-TAX, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(PAY-INCOME-TAX, (EQ (TURN-STAGE) VISIT))
ADDPRECONDITION(PAY-INCOME-TAX, (EQ (PLAYER-POSITION ?P) ?POS))
ADDPRECONDITION(PAY-INCOME-TAX, (EQ (POSITION-SPACE ?POS) INCOME-TAX))
ADDPRECONDITION(PAY-INCOME-TAX,
                  (FORALL (?SP – SPACE) (EQ (SPACE-TYPE ?SP) UTILITY) (NOT (EQ (SPACE-OWNER ?SP) ?P)))))
ADDEFFECT(PAY-INCOME-TAX, (DECREASE (PLAYER-CASH ?P) 200))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change Sometimes the decrease in player cash doesn't happen.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn.

Novelty Obviousness: Novel Transition Discernible Transitions in which the player does not have to pay are immediately recognizable.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until it lands on income tax and doesn't pay.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: We model this as an event novelty, because paying is not a choice that the agent makes, and the new PAY-INCOME-TAX event is not environmental, because the PLAYER-OWNED fluent is within the player's control.



Higher level novelties are not present: No higher levels exist.



*Simpler novelty is **technically** possible:* Technically, removing the position condition would still yield an environment novelty in which an additional tax exists, that happens every turn rather than at a specific spot. However, this doesn't match Monopoly intuition well. Might be an interesting consideration though.

Target Level: 8 - Events

Novelty Description:

alternate_buy_property_with_neighbour_tax

If a player wants to buy a property, the player will also have to pay fee to the players who own the left and right real estate properties. The fee is calculated by = left or right property's price/5. If left or right property is owned by itself or bank, then this fee is exempted.

Discussion:

After purchase, an extra fee is paid to neighbors equal to half of their property's price.

Assumed existing T-SAL action descriptions

```
(:ACTION PURCHASE
:PARAMETERS (?SPACE - SPACE)
:PERFORMER ?P
:PRECONDITION
  (AND
    (EQ (TURN-STAGE) VISIT)
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TITLE ?SPACE) ?TITLE)
    (NEQ (SPACE-TITLE ?SPACE) NOTITLE)
    (EQ (SPACE-OWNED ?SPACE) NOPLAYER)
    (< (TITLE-COST ?TITLE) (PLAYER-CASH ?P))
    (EQ (TITLE-RENT-UNIMPROVED ?TITLE) ?NEW-RENT)
    (EQ (TITLE-COST ?TITLE) ?COST)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?COST)
    (SET (SPACE-OWNED ?SPACE) ?P)
    (JUST-OBTAINED ?SPACE)
    (SET (SPACE-RENT ?SPACE) ?NEW-RENT)
    (SET (TURN-STAGE) POST_ROLL)
  )
)
```

Estimated T-SAL description of novel actions

```
(:EVENT PAY-NEIGHBOR-TAX
:PARAMETERS ()
:PRECONDITION
  (AND (EQ (CURRENT-PLAYER) ?P)
    (JUST-OBTAINED ?SPACE))
```

```

      (EQ (SPACE-NUM ?SPACE) ?POS)
      (EQ (SPACE-NUM ?NEXT-DOOR) ?POS2)
      (= (ABS (- ?POS ?POS2)) 1)
      (EQ (SPACE-OWNED ?NEXT-DOOR) ?NEIGHBOR)
      (NEQ (SPACE-OWNED ?NEXT-DOOR) NO-PLAYER)
      (EQ (SPACE-TITLE ?NEXT-DOOR) ?TITLE)
      (EQ (TITLE-COST ?TITLE) ?COST)
    )
  :EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (/ ?COST 2))
    (INCREASE (PLAYER-CASH ?NEIGHBOR) (/ ?COST 2))
  )
)

```

Estimated T-SAL Transformation Sequence:

```

ADDEVENT(PAY-NEIGHBOR-TAX)
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (JUST-OBTAINED ?SPACE))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (EQ (SPACE-NUM ?SPACE) ?POS))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (EQ (SPACE-NUM ?NEXT-DOOR) ?POS2))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (= (ABS (- ?POS ?POS2)) 1))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (EQ (SPACE-OWNED ?NEXT-DOOR) ?NEIGHBOR))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (NEQ (SPACE-OWNED ?NEXT-DOOR) NO-PLAYER))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (EQ (SPACE-TITLE ?NEXT-DOOR) ?TITLE))
ADDPRECONDITION(PAY-NEIGHBOR-TAX, (EQ (TITLE-COST ?TITLE) ?COST))
ADDEFFECT(PAY-NEIGHBOR-TAX, (DECREASE (PLAYER-CASH ?P) (/ ?COST 2)))
ADDEFFECT(PAY-NEIGHBOR-TAX, (INCREASE (PLAYER-CASH ?NEIGHBOR) (/ ?COST 2)))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Added Change Sometimes a purchase carries an extra money reduction.

Novelty Entry Determinism: Infrequent Nondeterministic: The proximate cause of purchase decisions is a nondeterministic roll. The agent's chance of landing on income tax on any given roll is never higher than 1 in 6 with standard dice; this could change in combination with other novelties.

Novelty Obviousness: Novel Transition Discernible Transitions in which the player loses extra cash are immediately recognizable.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until it purchases a property and pays extra.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

-  *Analyzed level matches intended level:* We naturally model this as an event novelty, because the extra tax is not always part of the PURCHASE action.
-  *Higher level novelties are not present:* No higher levels exist.
-  *Simpler novelty is **technically** possible:* You could technically remove the “next door” requirement, yielding a simpler novelty in which a player had to pay tax to all other owners on the board. However, this significantly changes the spirit.

Target Level: 8 - Events

Novelty Description:

alternate_pay_utility_rent

If a player is landing on owned utility (owned by others), this player will pay the utility rent to the bank instead of paying to the utility owner.

Assumed existing T-SAL action descriptions

```
(:EVENT PAY-UTILITY
:PARAMETERS ()
:PRECONDITION (AND (EQ (CURRENT-PLAYER) ?P)
  (EQ (TURN-STAGE) VISIT)
  (EQ (PLAYER-POSITION ?P) ?POS)
  (EQ (POSITION-SPACE ?POS) ?SPACE)
  (EQ (SPACE-TYPE ?SPACE) UTILITY)
  (NEQ (SPACE-OWNED ?SPACE) NOPLAYER)
  (EQ (SPACE-OWNED ?SPACE) ?OWNER)
  (EQ (UTILITY-MULTIPLIER) ?MULT)
)
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* ?MULT (+ (DIE1-VALUE) (DIE2-VALUE))))
    (INCREASE (PLAYER-CASH ?OWNER) (* ?MULT (+ (DIE1-VALUE) (DIE2-VALUE))))
    (SET (TURN-STAGE) POST_ROLL)
  )
)
```

Estimated T-SAL description of novel actions

```
(:EVENT PAY-UTILITY
:PARAMETERS ()
:PRECONDITION (AND (EQ (CURRENT-PLAYER) ?P)
  (EQ (TURN-STAGE) VISIT)
  (EQ (PLAYER-POSITION ?P) ?POS)
  (EQ (POSITION-SPACE ?POS) ?SPACE)
  (EQ (SPACE-TYPE ?SPACE) UTILITY)
  (NEQ (SPACE-OWNED ?SPACE) NOPLAYER)
  (EQ (SPACE-OWNED ?SPACE) ?OWNER)
  (EQ (UTILITY-MULTIPLIER) ?MULT)
)
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) (* ?MULT (+ (DIE1-VALUE) (DIE2-VALUE))))
    (INCREASE (PLAYER-CASH ?OWNER) (* ?MULT (+ (DIE1-VALUE) (DIE2-VALUE))))
    (SET (TURN-STAGE) POST_ROLL)
  )
)
```


)
)

Estimated T-SAL Transformation Sequence:

REMOVEEFFECT(PAY-UTILITY, (INCREASE (PLAYER-CASH ?OWNER) (* ?MULT (+ (DIE1-VALUE) (DIE2-VALUE)))))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change Owner of a utility does not receive rent money.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the possible effect of landing on a utility, where its owner does not collect rent.

Novelty Obviousness: Novel Transition Discernible Transitions in which a player doesn't gain expected cash are immediately recognizable.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on an owned utility and its owner does not collect.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This changes the effects of a non-environmental event, PAY-UTILITY.



Higher level novelties are not present: No higher levels exist.



Simpler novelty is not possible: Only one item in the transformation sequence.

Target Level: 8 - Events

Novelty Description:

calculate_rent_dynamic, rent_dynamic_perc' = 5

Rents on all properties not owned by the bank are updated every time a player passes go. The formula applied for house/hotel rents is: $1.1 * \text{original\_rent}$. Rent for unimproved real estate is $6 * \text{original\_rent}$. Rents for railroads and utilities are governed by the formula: $3.5 * \text{original\_rent}$. Note that the original rent is what the player would have paid in the default game.

Discussion:

Language seems to conflict, so assuming that rents are compounded, rather than simply being increased once and not changing again; $1.1 * \text{prior\_rent}$ rather than $1.1 * \text{original\_rent}$. Assuming that purchasing a house would resets the rent to the title value rather than maintaining the current multiplier. Since utilities do not have set rents, it's not clear how to make that work.

Assuming that rents are not directly observed except when paying; this may make the novelty harder to understand, since the rent may change several times between visits when rent is collected.

Estimated T-SAL description of novel actions

```
(:EVENT RENT-GOES-UP-IMPROVED
:PARAMETERS ()
:PRECONDITION
  (AND
    (JUST-PASSED-GO ?P)
    (EQ (SPACE-RENT ?SP) ?RENT)
    (NEQ (SPACE-OWNED ?SP) NO-PLAYER)
    (> (SPACE-HOUSES ?SP) 0)
  )
:EFFECT
  (AND
    (SET (SPACE-RENT ?SP) (* ?RENT 1.1))
  )
)
(:EVENT RENT-GOES-UP-UNIMPROVED
:PARAMETERS ()
:PRECONDITION
  (AND
    (JUST-PASSED-GO ?P)
    (EQ (SPACE-RENT ?SP) ?RENT)
    (NEQ (SPACE-OWNED ?SP) NO-PLAYER)
    (= (SPACE-HOUSES ?SP) 0)
  )
)
```

```

)
:EFFECT
  (AND
    (SET (SPACE-RENT ?SP) (* ?RENT 6))
  )
)
(:EVENT RENT-GOES-UP-RAILROAD-UTILITY
:PARAMETERS ()
:PRECONDITION
  (AND
    (JUST-PASSED-GO ?P)
    (EQ (SPACE-RENT ?SP) ?RENT)
    (NEQ (SPACE-OWNED ?SP) NO-PLAYER)
    (NEQ (SPACE-TYPE ?SP) PROPERTY)
    (NEQ (SPACE-TYPE ?SP) TAX)
  )
:EFFECT
  (AND
    (SET (SPACE-RENT ?SP) (* ?RENT 3.5))
  )
)

```

Estimated T-SAL Transformation Sequence:

```

ADDEVENT(RENT-GOES-UP-IMPROVED)
ADDPRECONDITION(RENT-GOES-UP-IMPROVED, (JUST-PASSED-GO ?P))
ADDPRECONDITION(RENT-GOES-UP-IMPROVED, (EQ (SPACE-RENT ?SP) ?RENT))
ADDPRECONDITION(RENT-GOES-UP-IMPROVED, (NEQ (SPACE-OWNED ?SP) NO-PLAYER))
ADDPRECONDITION(RENT-GOES-UP-IMPROVED, (> (SPACE-HOUSES ?SP) 0))
ADDEFFECT(RENT-GOES-UP-IMPROVED, (SET (SPACE-RENT ?SP) (* ?RENT 1.1)))
ADDEVENT(RENT-GOES-UP-UNIMPROVED)
ADDPRECONDITION(RENT-GOES-UP-UNIMPROVED, (JUST-PASSED-GO ?P))
ADDPRECONDITION(RENT-GOES-UP-UNIMPROVED, (EQ (SPACE-RENT ?SP) ?RENT))
ADDPRECONDITION(RENT-GOES-UP-UNIMPROVED, (NEQ (SPACE-OWNED ?SP) NO-PLAYER))
ADDPRECONDITION(RENT-GOES-UP-UNIMPROVED, (= (SPACE-HOUSES ?SP) 0))
ADDEFFECT(RENT-GOES-UP-UNIMPROVED, (SET (SPACE-RENT ?SP) (* ?RENT 6)))
ADDEVENT(RENT-GOES-UP-RAILROAD-UTILITY)
ADDPRECONDITION(RENT-GOES-UP-RAILROAD-UTILITY, (JUST-PASSED-GO ?P))
ADDPRECONDITION(RENT-GOES-UP-RAILROAD-UTILITY, (EQ (SPACE-RENT ?SP) ?RENT))
ADDPRECONDITION(RENT-GOES-UP-RAILROAD-UTILITY, (NEQ (SPACE-OWNED ?SP) NO-PLAYER))
ADDPRECONDITION(RENT-GOES-UP-RAILROAD-UTILITY, (NEQ (SPACE-TYPE ?SP) PROPERTY))
ADDPRECONDITION(RENT-GOES-UP-RAILROAD-UTILITY, (NEQ (SPACE-TYPE ?SP) TAX))
ADDEFFECT(RENT-GOES-UP-RAILROAD-UTILITY, (SET (SPACE-RENT ?SP) (* ?RENT 3.5)))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Size Rents increase in size.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of landing on an owned space and paying higher rent.

Novelty Obviousness: Novel Transition Discernible Transitions in which higher rents are paid are immediately recognizable.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on an owned space with higher rent.

Novel State Observation Activity: Passive: New rents will be observed during the natural course of the game, illuminating hidden rent state.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This creates non-environmental event, as players have indirect influence through purchase of properties.



Higher level novelties are not present: No higher levels exist.



*Simpler novelty **is possible**:* Three different changes, partially determined by improvements, is more complex than needed.

Possible remedy: The simplest version of this would have rent increase on all spaces at the same rate, whether owned or unowned.

Target Level: 8 - Events

Novelty Description:

alternate_move_player_on_landing_GO

If a player is landing on GO and also possesses full color set, then the player will be moved to the nearest monopolized color location after GO

Discussion:

Assuming move to that player's nearest color set.

Nearest monopolized color location is a somewhat complicated concept.

Estimated T-SAL description of novel actions

```
(:EVENT GO-ALT-MOVE
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TYPE ?SPACE) GO)
    (EQ (SPACE-OWNED ?NEW-SPACE) ?P)
    (EQ (SPACE-NUM ?NEW-SPACE) ?NEW-POS)
    (EQ (MONOPOLY-OWNED ?COLOR) ?P)
    (EQ (SPACE-COLOR ?NEW-SPACE) ?COLOR)
    (FORALL (?OTHER-SPACE – SPACE)
      (AND (EQ (SPACE-OWNED ?OTHER-SPACE) ?P) (EQ (SPACE-COLOR ?OTHER-SPACE) ?OTHER-COLOR)
        (EQ (MONOPOLY-OWNED ?OTHER-COLOR) ?P))
      (>= (SPACE-NUM ?OTHER-SPACE) ?NEW-POS))
  )
:EFFECT
  (AND
    (SET (PLAYER-POSITION ?P) ?NEW-POS)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDEVENT(GO-ALT-MOVE)
ADDPRECONDITION(GO-ALT-MOVE, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (TURN-STAGE) VISIT))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (PLAYER-POSITION ?P) ?POS))
```

```

ADDPRECONDITION(GO-ALT-MOVE, (EQ (POSITION-SPACE ?POS) ?SPACE))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (SPACE-TYPE ?SPACE) GO))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (SPACE-OWNED ?NEW-SPACE) ?P))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (SPACE-OWNED ?NEW-SPACE) ?NEW-POS))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (MONOPOLY-OWNED ?COLOR) ?P))
ADDPRECONDITION(GO-ALT-MOVE, (EQ (SPACE-COLOR ?NEW-SPACE) ?COLOR))
ADDPRECONDITION(GO-ALT-MOVE,
  (FORALL (?OTHER-SPACE – SPACE)
    (AND (EQ (SPACE-OWNED ?OTHER-SPACE) ?P) (EQ (SPACE-COLOR ?OTHER-SPACE) ?OTHER-COLOR)
      (EQ (MONOPOLY-OWNED ?OTHER-COLOR) ?P))
    (>= (SPACE-NUM ?OTHER-SPACE) ?NEW-POS)))
ADDEFFECT(GO-ALT-MOVE, (SET (PLAYER-POSITION ?P) ?NEW-POS))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Size Rents increase in size.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of landing on go, if they have a monopoly.

Novelty Obviousness: Novel Transition Discernible Transitions in which the player moves to a new place are immediately recognizable.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on go after obtaining a Monopoly.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This creates a non-environmental event, as players have indirect influence through obtaining properties and monopolies.
- ✓ *Higher level novelties are not present:* No higher levels exist.
- ✗ *Simpler novelty is possible:* Removal of multiple preconditions can be performed without reducing novelty level.

Possible remedy: The simplest version of this might be to have a player transition to their lowest numbered owned property.

Target Level: 8 - Events

Novelty Description:

alternate_tax_calc_based_on_property_ownership

If a player is landing on tax location, the tax is modified to the max rent of owned property, including real_estate, railroad and utility

Discussion:

Assuming this is new behavior of income tax and luxury tax.

Assuming that it's the max rent of that player's properties.

Estimated T-SAL description of novel actions

```
(:EVENT PAY-OWNERSHIP-TAX
:PARAMETERS ()
:PRECONDITION
  (AND
    (EQ (CURRENT-PLAYER) ?P)
    (EQ (TURN-STAGE) VISIT)
    (EQ (PLAYER-POSITION ?P) ?POS)
    (EQ (POSITION-SPACE ?POS) ?SPACE)
    (EQ (SPACE-TYPE ?SPACE) TAX)
    (EQ (SPACE-OWNED ?HIGH-RENT-SPOT) ?P)
    (EQ (SPACE-RENT ?HIGH-RENT-SPOT) ?RENT)
    (FORALL (?OTHER-SPACE — SPACE)
      (AND (EQ (SPACE-OWNED ?OTHER-SPACE) ?P))
      (<= (SPACE-RENT ?OTHER-SPACE) ?RENT))
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) ?RENT)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDFLUENTVALUE(SPACE-RENT, [INCOME-TAX], 0)
ADDFLUENTVALUE(SPACE-RENT, [LUXURY-TAX], 0)
ADDEVENT(PAY-OWNERSHIP-TAX)
ADDPRECONDITION(PAY-OWNERSHIP-TAX, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(PAY-OWNERSHIP-TAX, (EQ (TURN-STAGE) VISIT))
ADDPRECONDITION(PAY-OWNERSHIP-TAX, (EQ (PLAYER-POSITION ?P) ?POS))
ADDPRECONDITION(PAY-OWNERSHIP-TAX, (EQ (POSITION-SPACE ?POS) ?SPACE))
```

```

ADDPRECONDITION(PAY-OWNERSHIP-TAX, (EQ (SPACE-TYPE ?SPACE) TAX))
ADDPRECONDITION(PAY-OWNERSHIP-TAX, (EQ (SPACE-OWNED ?HIGH-RENT-SPOT) ?RENT))
ADDPRECONDITION(PAY-OWNERSHIP-TAX,
  (FORALL (?OTHER-SPACE — SPACE)
    (EQ (SPACE-OWNED ?OTHER-SPACE) ?P)
    (<= (SPACE-RENT ?OTHER-SPACE) ?RENT)))
ADDEFFECT(PAY-OWNERSHIP-TAX, (DECREASE (PLAYER-CASH ?P) ?RENT))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Size Tax amounts change.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of landing on a tax spot.

Novelty Obviousness: Novel Transition Discernible Changes to the tax amount are immediately discernible.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on a tax spot.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This creates a non-environmental event, as players have indirect influence through purchasing and improving properties.



Higher level novelties are not present: No higher levels exist.



*Simpler novelty is **technically** possible:* In our framework, it would be easier to express paying tax equal to the cost of all properties, but this would not necessarily be easier for most agents to recognize.

Target Level: 8 - Events

Novelty Description:

inflation_increment, 100

Every time a player passes go, for every colored asset which still owned by bank and price <1500, the price increases by an amount that is equal to $[100 * \text{color\_count}]$, where color_count is the number of properties of that color not owned by the bank.

Discussion:

Assuming that changes to costs of properties are not immediately communicated. This makes the novelty much harder to understand, though, since the property's price may change multiple times before it's observed.

Estimated T-SAL description of novel actions

```
(:EVENT INFLATION-INCREASES-PRICE
:PARAMETERS ()
:PRECONDITION
  (AND
    (JUST-PASSED-GO ?P)
    (EQ (SPACE-OWNED ?UNOWNED-SPACE) NO-PLAYER)
    (EQ (SPACE-COLOR ?UNOWNED-SPACE) ?COLOR)
    (EQ (SPACE-TITLE ?UNOWNED-SPACE) ?TITLE)
    (< (TITLE-COST ?UNOWNED-SPACE) 1500)
    (SUM ?INFLATION (?COLOR-SPACE - SPACE)
      (EQ (SPACE-COLOR ?COLOR-SPACE) ?COLOR)
      100)
  )
:EFFECT
  (AND
    (INCREASE (TITLE-COST ?TITLE) ?INFLATION)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDEVENT(INFLATION-INCREASES-PRICE)
ADDPRECONDITION(INFLATION-INCREASES-PRICE, (JUST-PASSED-GO ?P))
ADDPRECONDITION(INFLATION-INCREASES-PRICE, (EQ (SPACE-OWNED ?UNOWNED-SPACE) NO-PLAYER))
ADDPRECONDITION(INFLATION-INCREASES-PRICE, (EQ (SPACE-COLOR ?UNOWNED-SPACE) ?COLOR))
ADDPRECONDITION(INFLATION-INCREASES-PRICE, (EQ (SPACE-TITLE ?UNOWNED-SPACE) ?TITLE))
ADDPRECONDITION(INFLATION-INCREASES-PRICE, (< (TITLE-COST ?UNOWNED-SPACE) 1500))
ADDPRECONDITION(INFLATION-INCREASES-PRICE,
  (SUM ?INFLATION (?COLOR-SPACE - SPACE) (EQ (SPACE-COLOR ?COLOR-SPACE) ?COLOR) 100))
ADDEFFECT(INFLATION-INCREASES-PRICE, (INCREASE (TITLE-COST ?TITLE) ?INFLATION))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in Change Size Price paid during purchases changes.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of passing go.

Novelty Obviousness: Novel Transition Discernible Changes to the price are discernible. This comes well after the inflation starts occurring, of course

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on a tax spot.

Novel State Observation Activity: Active: Player may need to attempt to purchase a property in order to determine that the price has risen.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This creates a non-environmental event, as players have indirect influence through purchasing properties.



Higher level novelties are not present: No higher levels exist.



*Simpler novelty **is possible**:* Removing the upper-bound (1500) on property costs still leaves an event novelty.

Target Level: 8 - Events

Novelty Description:

alternate_railroads_functionality

If a player lands on owned railroad, then the player will be moved to 7 locations behind.

Discussion:

Assuming that it doesn't matter who owns the railroad for purposes of moving players.

Assuming that rent is still collected.

Estimated T-SAL description of novel actions

```
(:EVENT RAILROAD-MOVES-BACKWARDS
:PARAMETERS ()
:PRECONDITION
  (AND (EQ (CURRENT-PLAYER) ?P)
        (EQ (TURN-STAGE) VISIT)
        (EQ (PLAYER-POSITION ?P) ?POS)
        (EQ (POSITION-SPACE ?POS) ?SPACE)
        (EQ (SPACE-TYPE ?SPACE) RAILROAD)
        (NEQ (SPACE-OWNED ?SPACE) NO-PLAYER)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-POSITION ?P) 7)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDEVENT(RAILROAD-MOVES-BACKWARDS)
ADDPRECONDITION(RAILROAD-MOVES-BACKWARDS, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(RAILROAD-MOVES-BACKWARDS, (EQ (TURN-STAGE) VISIT))
ADDPRECONDITION(RAILROAD-MOVES-BACKWARDS, (EQ (PLAYER-POSITION ?P) ?POS))
ADDPRECONDITION(RAILROAD-MOVES-BACKWARDS, (EQ (POSITION-SPACE ?POS) ?SPACE))
ADDPRECONDITION(RAILROAD-MOVES-BACKWARDS, (EQ (SPACE-TYPE ?SPACE) RAILROAD))
ADDPRECONDITION(RAILROAD-MOVES-BACKWARDS, (NEQ (SPACE-OWNED ?SPACE) NO-PLAYER))
ADDEFFECT(RAILROAD-MOVES-BACKWARDS, (DECREASE (PLAYER-POSITION ?P) 7))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: New Change Position changes at a new time.

Novelty Entry Determinism: Deterministic: Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of landing on an owned railroad.

Novelty Obviousness: Novel Transition Discernible Changes to position are discernible.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on a tax spot.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This creates a non-environmental event, as players have indirect influence through purchasing and improving properties.
- ✓ *Higher level novelties are not present:* No higher levels exist.
- ✓ *Simpler novelty is not possible:* Nothing can be removed while maintaining an event novelty.

Target Level: 8 - Events

Novelty Description:

last_loc_monopoly_tax

If a player owns the color set and is also landing on last location in this color set, then the player have to pay 200 dollars to the bank.

Discussion:

“Last location in my color set” is a complex concept

Estimated T-SAL description of novel actions

```
(:EVENT LAST-LOCATION-TAX
:PARAMETERS ()
:PRECONDITION
  (AND (EQ (CURRENT-PLAYER) ?P)
        (EQ (TURN-STAGE) VISIT)
        (EQ (PLAYER-POSITION ?P) ?POS)
        (EQ (POSITION-SPACE ?POS) ?SPACE)
        (EQ (SPACE-OWNED ?SPACE) ?P)
        (EQ (SPACE-COLOR ?SPACE) ?COLOR)
        (EQ (MONOPOLY-OWNED ?COLOR) ?P)
        (FORALL (?COLOR-SPACE — SPACE)
          (EQ (SPACE-COLOR ?SPACE) ?COLOR)
          (<= (SPACE-NUM ?SPACE) ?POS)))
)
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) 200)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDEVENT(LAST-LOCATION-TAX)
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (TURN-STAGE) VISIT))
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (PLAYER-POSITION ?P) ?POS))
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (POSITION-SPACE ?POS) ?SPACE))
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (SPACE-OWNED ?SPACE) ?P))
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (SPACE-COLOR ?SPACE) ?COLOR))
ADDPRECONDITION(LAST-LOCATION-TAX, (EQ (MONOPOLY-OWNED ?COLOR) ?P))
ADDPRECONDITION(LAST-LOCATION-TAX,
  (FORALL (?COLOR-SPACE — SPACE) (EQ (SPACE-COLOR ?SPACE) ?COLOR) (<= (SPACE-NUM ?SPACE) ?POS)))
ADDEFFECT(LAST-LOCATION-TAX, (DECREASE (PLAYER-CASH ?P) 200))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: New Change Cash changes at a new time.

Novelty Entry Determinism: Deterministic Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of landing on last spot of their monopoly.

Novelty Obviousness: Novel Transition Discernible Changes to cash are discernible.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on last location of their monopoly.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: This creates a non-environmental event, as players have indirect influence through obtaining monopolies.



Higher level novelties are not present: No higher levels exist.



*Simpler novelty **is possible**:* There are two conditions for this event: last spot and owned monopoly. Remove one, and there's still a viable event novelty.

Possible remedy: The simplest version likely taxes a player whenever they land on owned properties.

Target Level: 8 - Events

Novelty Description:

alternate_player_movement

If a player is landing on the location where other players already arrived, then the latter arrived player have to pay 400 dollars to the first arrived player.

Assumption:

New player pays 400 to every player already on a spot, not whoever arrived first

Estimated T-SAL description of novel actions

```
(:EVENT SECOND-VISITOR-TAX
:PARAMETERS ()
:PRECONDITION
  (AND (EQ (CURRENT-PLAYER) ?P)
        (EQ (TURN-STAGE) VISIT)
        (EQ (PLAYER-POSITION ?P) ?POS)
        (EQ (PLAYER-POSITION ?OP) ?POS)
        (NEQ (CURRENT-PLAYER) ?OP)
  )
:EFFECT
  (AND
    (DECREASE (PLAYER-CASH ?P) 400)
    (INCREASE (PLAYER-CASH ?OP) 400)
  )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDEVENT(SECOND-VISITOR-TAX)
ADDPRECONDITION(SECOND-VISITOR-TAX, (EQ (CURRENT-PLAYER) ?P))
ADDPRECONDITION(SECOND-VISITOR-TAX, (EQ (TURN-STAGE) VISIT))
ADDPRECONDITION(SECOND-VISITOR-TAX, (EQ (PLAYER-POSITION ?P) ?POS))
ADDPRECONDITION(SECOND-VISITOR-TAX, (EQ (PLAYER-POSITION ?OP) ?POS))
ADDPRECONDITION(SECOND-VISITOR-TAX, (NEQ (CURRENT-PLAYER) ?OP))
ADDEFFECT(SECOND-VISITOR-TAX, (DECREASE (PLAYER-CASH ?P) 400))
ADDEFFECT(SECOND-VISITOR-TAX, (INCREASE (PLAYER-CASH ?OP) 400))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: New Change Additional cash payments made.

Novelty Entry Determinism: Agent-Caused Entry into the novelty region happens when any player decides to pass turn. Then another player has the option to roll with the effect of landing on last spot of their monopoly.

Novelty Obviousness: Novel Transition Discernible Changes to cash are discernible.

Novelty Observation Determinism: Deterministic All new dynamics are observed deterministically, as Monopoly is essentially fully observable.

Novelty Observation Localization: Nearby Agent will not observe novelty until a player lands on top of another player.

Novel State Observation Activity: N/A: There are no novel states.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis

-  *Analyzed level matches intended level:* This creates an environmental event, as players have no influence on where they land. **Possible remedy:** If the event were contingent on how much cash the player has (e.g., you have to pay if you have more cash than the other player), that would be within player control and would become a non-environmental event.
-  *Higher level novelties are not present:* No higher levels exist.
-  *Simpler novelty is not possible:* Technically, the money could disappear instead, which would remove an effect while not changing the novelty level. However, this is a small deviation.

Monopoly Combined Level 8 Novelties

Target Level: 8 - Events

Novelty Description:

Combinations

Discussion:

All of the Monopoly full novelties are combinations of the above primitives. This section discusses what that means for the overall novelty.

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only – none of the primitive event novelties described affect the agent logic.

Transition Novelty Type: Multiple When multiple novelties are put together, multiple transition novelty types apply.

Novelty Entry Determinism: Infrequent Nondeterministic/Deterministic: Novelties that affect purchasing (avenue_rule_change and neighbor_tax) are infrequent nondeterministic, because players receive these opportunities nondeterministically. If either is involved in a combination, the nondeterministic answer overrides the deterministic answer of the other novelties.

Novelty Obviousness: Novel Transition Discernible All event novelties have discernible transitions.

Novelty Observation Determinism: Deterministic All observations in Monopoly are deterministic.

Novelty Observation Localization: Nearby All event novelties in Monopoly are nearby; observations are received only after the event transition has occurred.

Novel State Observation Activity: N/A / Passive / Active: Novel states occur in novelties that change property rents and costs, as these are state information that can be hidden up until a purchase or rent payment. (Unless these assumptions are incorrect.) Other events do not introduce novel states, only novel transitions. If active and passive primitives are combined the entire novelty is active.

Novelty Frequency: Less Frequent than Observation: Always true in Monopoly.

Hierarchy Level Analysis



Analyzed level matches intended level: All primitives are event novelties, so event novelty exists in all combinations.



Higher level novelties are not present: No higher levels exist.



Simpler novelty is possible: In any combination of primitives, one or more primitives can be removed, so this condition is not met by the combinations.

Novelty Analysis for university of Texas at Dallas Novelty Generation Team

Matthew Molineaux – Parallax Advanced Research – matthew.molineaux@parallaxresearch.org
Prepared with support from PacMar Technologies

October 13, 2022



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

Analysis Purpose and Method

We are providing feedback to help your decision-making process on the SAIL-ON program as a TA2, not attempting to call your decisions into question. Our primary goal is to provide you with descriptions of the novelty dimensions and estimated difficulties of them, for you to consider; and to provide you with our assessments of how well they meet the novelty hierarchy level as we have formalized it in our theory (see Transformation and Hierarchy Formalization document).

For each novelty below, we describe the following elements:

- Target Level: The novelty hierarchy level you targeted
- Novelty Description: Your description of the novelty
- Assumptions: Assumptions we made about the novelty that may or may not be correct.
- Discussion: We use this section to provide context for how we look at the novelty and how we model it.
- Estimated T-SAL Environment Model Elements: Excerpts from a hypothetical T-SAL model of the modified and/or original environment that describe the novelty.
- Estimated T-SAL Transformation Sequence: Formal description of the novelty as a sequence of transformations between initial environment model and scenario generator to modified environment model and scenario generator; as described in the Transformation and Hierarchy Formalization document
- Novelty Dimensions: Assignment of novelty characteristics along eight environment transformation dimensions described in our Environment Transformation Framework theory, and explanation of each assignment
- Hierarchy Level Analysis: An analysis of whether the described novelty fits the definitions of hierarchy levels in the SAIL-ON program, based on our formal theory describing each.

Our assessments of novelty dimension values and hierarchy level analysis are based on the T-SAL language (itself primarily based on PDDL+) and transformations to environments described in that language. As such, the estimated transformation sequence is the formal basis for our determinations, and it's based on the estimated environment model elements required to describe a novelty. In many cases, multiple transformation sequences could create different environment models that would explain the observations generated equally well. When we see this as being possible, we have noted it.

The hierarchy level analysis for each novelty answers three questions:

1. Does the novelty, as described, clearly have the characteristics of the stated level as described in our theory? Might it be described at a lower one?
2. Does the novelty **also** have characteristics from a higher novelty level, which might make it more difficult than intended?

3. Is the novelty as simple as possible? This is related to PM Ted Senator’s request that only a “single” novelty be presented at each point. Our transformation sequences make clear that it is usually not possible to describe a novelty with only a single textual change to a model, so we ask instead whether a simpler transformation sequence can be constructed, by removing one or more existing elements, that still meets the requirements of the target novelty level. There may be circumstances in which meeting this criterion is impractical.

These questions have the following possible responses:

Q	Icon	Response	Meaning
1		<i>Analyzed level matches intended level</i>	Our theory of hierarchy levels treats the level of the novelty as the same one you provided.
		<i>Could be modelled at lower level</i>	We found a model for the novelty as described that matches the provided level in our theory of hierarchy levels, but we also found an alternative model at a lower novelty level. This might confuse some agents.
		<i>Analyzed level does not match intended level</i>	We did not find a model for the novelty that matches your provided level.
2		<i>Higher level novelties not present</i>	The transformation sequence found does not also meet the conditions for a higher level novelty than that stated.
		<i>Higher level novelties are present</i>	The transformation sequence found meets the conditions for a higher level novelty than that stated, in addition to conditions for a lower level.
3		<i>Simpler novelty not possible</i>	There is no way to remove a transformation from the T-SAL transformation sequence that models this novelty and arrive at a sequence from the described level.
		<i>Simpler novelty is possible</i>	One or more T-SAL transformations could be removed from this sequence, and the resulting sequence would still be a legal novelty at the intended level. (That novelty might be a nuisance novelty, though.)

In addition to the icon and response, we describe the rationale for each response, and provide a possible remedy, if you are interested in pursuing it, for “non-green” responses. There’s nothing binding about these, we’re just trying to provide the best feedback we can; please ignore whatever’s not helpful to you.

POGO

Target Level: 6 - Environment

Novelty Description:

- 1) Class: Wind
 - a. Impact: If a player moves into Wind, it has a X% chance to move an additional amount in the direction of the wind. The Wind can always be in the same direction.
 - b. Diffuse (random single block locations)
 - c. Zones (large areas of all wind that can be moved through or avoided)

Discussion:

Note that if this does not affect the pogoist, it looks like an action or event novelty more than an environmental one.

Assumptions:

- Discrete time (happens every time player moves)
- Wind exerts force on player only
- Agent can't see wind.
- Wind affects pogoist.

Assumed existing T-SAL descriptions

```
(:action move
:parameters (?direction - vector)
:performer ?ag
:precondition (and (can-move ?ag))
:effect (and (just-moved ?ag ?direction) (increase (position ?ag) ?direction))
)
```

```
(:event move-ends
:parameters ()
:precondition (and (just-moved ?ag ?direction))
:effect (and (not (just-moved ?ag ?direction)) )
)
```

Estimated T-SAL description of novel event

```
(:event wind-moves-player
:parameters (?ag – agent)
:probability ?prob
:precondition (and (just-moved ?ag ?direction) (eq (position ?ag) ?pos) (in-wind ?pos))
```

```
(eq (wind-vector) ?vec) (eq (wind-probability) ?prob)
:effect (and (increase (position ?player) ?vec))
)
```

Estimated T-SAL Transformation Sequence:

```
ADDFLUENT(in-wind)
ADDFLUENT(wind-vector)
ADDEVENT(wind-moves-player)
CHANGEVENTPROBABILITY(wind-moves-player, ?prob)
ADDPRECONDITION(wind-moves-player, "(just-moved ?player ?direction)")
ADDPRECONDITION(wind-moves-player, "(eq (position ?player) ?pos)")
ADDPRECONDITION(wind-moves-player, "(in-wind ?pos)")
ADDPRECONDITION(wind-moves-player, "(eq (wind-vector) ?vec)")
ADDPRECONDITION(wind-moves-player, "(eq (wind-probability) ?prob)")
ADDEFFECT(wind-moves-player, "(increase (position ?player) ?vec)")
ADDRELATIONGENERATOR("wind-vector", NRELATIONDRAWS([], Vectors, 1))
ADDRELATIONGENERATOR("wind-probability", NRELATIONDRAWS([],
UNIFORMDISTRIBUTION(0/EasyMax/MediumMax, EasyMax/MediumMax/HardMax), 1)
ADDRELATIONGENERATOR("in-wind", NRELATIONDRAWS([AllPositions], CONSTANTFUNCTION(TRUE),
EasyCount/MediumCount/HardCount))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: New Change, Increase/Decrease in Change Value Objects movements will be greater/smaller non-deterministically. Depending on representation, changes in X or Y might be introduced where only changes in the other were expected.

Novelty Entry Determinism: Agent-Caused Entry occurs when entering a windy location, which is under agent control.

Novelty Obviousness: Novel Transition Discernible Different changes in position should be observed immediately. (**Alien Observation** if wind is observed directly.)

Novelty Observation Determinism: None Assuming the wind is not observed directly, so there's no change.

Novelty Observation Localization: Nearby Agent always encounters novel transition first.

Novel State Observation Activity: Active: Due to nondeterministic wind, states with no influence on agent due to wind may look identical to states with influence on agent due to wind. Repeated movements from each possible position would be necessary to determine where wind exists.

Novelty Frequency: Less Frequent than Observation: Wind affects agent only when it moves (discrete time).

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is an environmental novelty, as it creates generators for three environmental fluents (wind-vector, wind-probability, and in-wind).
- ✓ *Higher level novelties **are** present:* This counts as an event novelty, because the agent has partial control over the novelty (by moving its position), but it affects the pogoist as well, so it's in the spirit of "affects everything identically". This is a borderline case.

Possible remedy: If the wind blows other things instead of the agent, that the agent does not normally affect the position of, it would be a clearly environmental effect according to our model. Maybe tumbleweeds containing resources?

- ✗ *Simpler novelty **is possible**:* Three fluents are added to the environment; only one is necessary.) **Possible remedy:** If you wish to make this simpler to model, consider making wind constant across the environment; this would be a simpler, irreducible version of the novelty.

Target Level: 6 - Environment

Novelty Description:

- 2) Class: Ice
 - a. Impact: If a player moves onto an ice block it gets frozen and can no longer move, ending the task. The movement is still a success, but all future movement actions are failures with a message about the ice. This is effectively a trap that must be avoided.
 - b. Diffuse
 - c. Zones

Assumptions:

- Agent can't tell move is impossible until it tries to move again.
- Ice was not previously present in environment.
- Agent can see ice
- Ice looks different from anything previously seen.
- Ice affects pogoist.

Discussion:

Note that if this does not affect the pogoist, it looks like an action or object novelty more than an environmental novelty.

It would seem more environmental (rather than object) if there were a ubiquitous chill in the air.

Estimated T-SAL description of novel event

```
(:event ice-freezes-player
:parameters (?player – agent)
:precondition (and (just-moved ?player ?direction) (eq (position ?player) ?pos) (ice-at ?pos))
:effect (and (not (can-move ?player))))
)
```

Estimated T-SAL Transformations:

```
ADDFLUENT(ice-at)
ADDPRECONDITION(ice-freezes-player, "(just-moved ?player ?direction)")
ADDPRECONDITION(ice-freezes-player, "(eq (position ?player) ?pos)")
ADDPRECONDITION(ice-freezes-player, "(ice-at ?pos)")
ADDEFFECT(ice-freezes-player, "(not (can-move ?player))")
ADDRELATIONGENERATOR("ice-at", NRELATIONDRAWS([AllPositions], CONSTANTFUNCTION(TRUE),
EasyCount/MediumCount/HardCount))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Change Movement no longer occurs away from spots where it used to.

Novelty Entry Determinism: Agent-Caused Entry occurs when entering an icy location, which is under agent control.

Novelty Obviousness: Alien Observation when ice is observed.

Novelty Observation Determinism: Deterministic/Frequent Nondeterministic Ice can be observed from a ways away, but camera might be essentially nondeterministic from certain angles. Mostly deterministic.

Novelty Observation Localization: Distant Agent sees ice before getting stuck.

Novel State Observation Activity: Active: Since ice might not be in original room, agent will need to move to first encounter ice, and won't be able to distinguish states with and without ice until it does.

Novelty Frequency: Less Frequent than Observation: Ice affects agent only once, when it gets stuck.

Hierarchy Level Analysis



Analyzed level matches intended level: This is an environmental novelty, as it creates a generator for an environmental fluent (ice-at) as well as an environmental event (ice-freezes-player).



*Higher level novelties **are** present:* This counts as an event novelty, because the agent has partial control over the "ice-freezes-player" event (by moving its position), but it affects the pogoist as well, so it's in the spirit of "affects everything identically". This is a borderline case.

Possible remedy: If the ice were to freeze resources rather than the player, so that the agent had to break it to continue, the agent's position would not be directly involved in the event.



Simpler novelty is not possible: The transformation sequence can not be reduced without changing novelty level.

Target Level: 6 - Environment

Novelty Description:

- 3) Class: Fire
 - a. Impact: If a player moves onto a fire block, they either die or experience a massive negative Cost.
 - b. Diffuse
 - c. Zones

Assumptions:

- Fire was not previously present in environment.
- Agent can see fire
- Fire looks different from anything previously seen.
- Fire affects pogoist.
- Cost for touching fire.

Discussion:

Dying is a lot like never moving again, I'm suggesting leaning towards a cost. However, cost won't be observable when pogoist does it, making this harder to learn about – it won't look any different from an action or event novelty to a system. Maybe it would be better if things in the inventory (e.g. wood) burned?

Estimated T-SAL description of novel event

```
(:event fire-burns-player
:parameters (?player – agent)
:precondition (and (just-moved ?player ?direction) (eq (object-position ?player) ?pos)
                 (fire-at ?pos))
:effect (and (increase (cost) 10000))
)
```

Estimated T-SAL Transformations:

```
ADDFLUENT(fire-at)
ADDPRECONDITION(fire-burns-player, "(just-moved ?player ?direction)")
ADDPRECONDITION(fire-burns-player, "(eq (position ?player) ?pos)")
ADDPRECONDITION(fire-burns-player, "(fire-at ?pos)")
ADDEFFECT(fire-burns-player, "(increase (cost) 10000)")
ADDRELATIONGENERATOR("fire-at", NRELATIONDRAWS([AllPositions], CONSTANTFUNCTION(TRUE),
EasyCount/MediumCount/HardCount))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase/Decrease in change size Change in cost is much higher for certain moves.

Novelty Entry Determinism: Agent-Caused Entry occurs when entering a fiery location, which is under agent control.

Novelty Obviousness: Alien Observation When fire is observed.

Novelty Observation Determinism: Deterministic/Frequent Nondeterministic Fire can be observed from a distance, but camera might be somewhat nondeterministic from certain angles and far distances. Mostly deterministic.

Novelty Observation Localization: Distant Agent sees fire before getting burned.

Novel State Observation Activity: Active: Since fire might not be in original room, agent will need to move to first encounter fire, and won't be able to distinguish states with and without fire until it does.

Novelty Frequency: Less Frequent than Observation: Fire affects agent only when it moves.

Hierarchy Level Analysis

✓ *Analyzed level matches intended level:* This is an environmental novelty, as it creates a generator for an environmental fluent (fire-at) as well as an environmental event (fire-burns-player).

✓ *Higher level novelties **are** present:* This counts as an event novelty, because the agent has partial control over the novelty (by moving its position), but it affects the pogoist as well, so it's in the spirit of "affects everything identically". This is a borderline case.

Possible remedy: Consider a more environmental effect in which the fire spreads outside of the agent's control and burns things other than the player whose position are outside the player's control (e.g., trees).

✓ *Simpler novelty is not possible:* The transformation sequence cannot be reduced without changing novelty level.

Target Level: 6 - Environment

Novelty Description:

- 4) Class: Snow
 - a. When you move or TP from an Air block with Snow, you sometimes fail to move and have to try again.
 - b. Diffuse
 - c. Zones

Assumptions:

- Snow was not previously present in environment.
- Agent can see snow.
- Snow looks different from anything previously seen.
- Snow affects pogoist.
- Cost for touching snow.

Discussion:

Failing to move is somewhere in between dying and wind maybe? A lot of movement-based ideas.

Estimated T-SAL description of modified action

```
(:action move
:parameters (?direction - vector)
:performer ?ag
:precondition (and (can-move ?ag) ) (eq (position ?player) ?pos))
:effect (and (just-moved ?ag ?direction) (increase (position ?ag) ?direction))
)
```

Note: Red, struck out text indicates where an effect or precondition is removed.

Estimated T-SAL description of novel event

```
(:event snow-move-happens
:parameters ()
:probability ?prob
:precondition (and (just-moved ?ag ?direction) (eq (position ?player) ?pos) (snow-at ?pos)
                  (eq (snow-move-chance) ?prob))
:effect (and (increase (position ?ag) ?direction))
)
```

```
(:event nonsnow-move-happens
:parameters ()
:precondition (and (just-moved ?ag ?direction) (eq (position ?player) ?pos))
```

```

                (not (snow-at ?pos)))
:effect (and (increase (position ?ag) ?direction) )
)

```

Estimated T-SAL Transformations:

```

ADDFLUENT(snow-at)
REMOVEEFFECT(move, "(increase (position ?ag) ?direction)")
ADDFLUENT(snow-move-chance)
ADDEVENT(snow-move-happens)
ADDEVENT(nonsnow-move-happens)
CHANGEVENTPROBABILITY(snow-move-happens, ?prob)
ADDPRECONDITION(snow-move-happens, "(just-moved ?player ?direction)")
ADDPRECONDITION(snow-move-happens, "(eq (position ?player) ?pos)")
ADDPRECONDITION(snow-move-happens, "(snow-at ?pos)")
ADDPRECONDITION(snow-move-happens, "(eq (snow-move-chance) ?prob)")
ADDEFFECT(snow-move-happens, "(increase (position ?ag) ?direction)")
ADDPRECONDITION(nonsnow-move-happens, "(just-moved ?player ?direction)")
ADDPRECONDITION(nonsnow-move-happens, "(eq (position ?player) ?pos)")
ADDPRECONDITION(nonsnow-move-happens, "(snow-at ?pos)")
ADDEFFECT(nonsnow-move-happens, "(increase (position ?ag) ?direction)")
ADDRELATIONGENERATOR("snow-at", NRELATIONDRAWS([AllPositions], CONSTANTFUNCTION(TRUE),
EasyCount/MediumCount/HardCount))

```

Alternate contingent-effect based description of modified action

```

(:action move
:parameters (?direction - vector)
:performer ?ag
:precondition (and (can-move ?ag) ) (eq (position ?player) ?pos))
:effect (and (just-moved ?ag ?direction) (increase (position ?ag) ?direction)
            (when (snow-at ?pos) (increase (position ?ag) ?direction))
            (when (not (snow-at ?pos)) (increase (position ?ag) ?direction)) 0.5)
)

```

Note: Red, struck out text indicates where an effect or precondition is removed.

Alternate contingent-effect-based Transformations:

```

ADDFLUENT(snow-at)
REMOVEEFFECT(move, "(increase (position ?ag) ?direction)")
ADDEFFECT(move, "(when (snow-at ?pos) (increase (position ?ag) ?direction))")
ADDEFFECT(move, "(when (not (snow-at ?pos)) (increase (position ?ag) ?direction)) 0.5")
ADDRELATIONGENERATOR("snow-at", NRELATIONDRAWS([AllPositions], CONSTANTFUNCTION(TRUE),
EasyCount/MediumCount/HardCount))

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Missing Changes Sometimes, a change in position does not occur

Novelty Entry Determinism: Agent-Caused Entry occurs when entering a snowy location, which is under agent control. (Unless agent starts in snow, then **None**).

Novelty Obviousness: Alien Observation When snow is observed.

Novelty Observation Determinism: Deterministic/Frequent Nondeterministic Snow can be observed from a ways away, but camera might be somewhat nondeterministic from certain angles and far distances. Mostly deterministic.

Novelty Observation Localization: Distant Agent sees snow before trying to move.

Novel State Observation Activity: Active: Since snow might not be in original room, agent will need to move to first encounter snow, and won't be able to distinguish states with and without snow until it does.

Novelty Frequency: Less Frequent than Observation: Snow affects agent only when it moves.

Hierarchy Level Analysis



Analyzed level matches intended level: This is an environmental novelty, as it creates a generator for an environmental fluent (snow-at).



Higher level novelties are present: This counts as an event novelty, because the agent has partial control over the novelty (by moving its position), but it affects the pogoist as well, so it's in the spirit of "affects everything identically". This is a borderline case.

Possible remedy: If the snow goes on existing objects and hides things (such as resources on the ground and block types), that affects the observation of blocks equally in a way that the agent can't control. This would be a more environmental effect according to our model.



Simpler novelty is possible: While the T-SAL based transformation sequence looks complicated, this is rendered much simpler using a contingent-effects-based representation. It is, however, still possible to make a simpler novelty by simply removing the "when snow sometimes move" contingent effect. This renders it effectively equivalent to the "ice" novelty. Essentially, the agent needs to learn both that snow affects movement AND that it sometimes doesn't, which may appear like separate learning problems.

Possible remedy: It may be useful to model the effect of snow on actions other than movement, such as collecting things. Slowing this down at a constant rate would be a single effect.

Target Level: 6 - Environment

Novelty Description:

- 5) Attributes: World Size
 - a. Impact: The size ~~and/or density~~ of objects changes
 - b. More objects
 - i. Arena gets bigger and more objects appear
 - c. ~~Same # of objects~~
 - i. ~~Arena gets smaller/bigger and object # stays the same~~

Note: Struck out text from Eric's novelty description indicates material not covered in this section.

Assumptions:

- Objects end up being made of more blocks, so are bigger relative to player
- Block is a unit of distance.
- Size is indicated by number of blocks.

Discussion:

Size only for this one; density handled separately.

Estimated Original T-SAL generators:

ADDVALUEGENERATOR(XLocations, INTEGERSEQUENCE(1, 1000))

ADDVALUEGENERATOR(YLocations, INTEGERSEQUENCE(1, 1000))

ADDVALUEGENERATOR(BlockTypes, DRAWFROMOBJECTSET(NEWSET("wood", "metal", "grass", "tree")))

ADDERELATIONGENERATOR("block-type-at", SPECIALDISTRIBUTION([XLocations, YLocations, BlockTypes]))

Estimated T-SAL Transformations:

ADDVALUEGENERATOR(XLocations, INTEGERSEQUENCE(1, *)) (larger or smaller)

ADDVALUEGENERATOR(YLocations, INTEGERSEQUENCE(1, *)) (larger or smaller)

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Added or Missing changes Moves that were previously blocked by arena edges no longer are (larger), or are blocked by closer ones (smaller).

Novelty Entry Determinism: None Novelty is present from initial state.

Novelty Obviousness: Alien Observation Different positions of walls will lead to observations outside the space possible in the original environment, whether the arena is smaller or larger.

Novelty Observation Determinism: Deterministic Should be apparent immediately that walls are closer or farther away (or, with a change in camera perspective, the blocks nearby seem smaller).

Novelty Observation Localization: Distant Agent sees new distances before being blocked or allowed through by them.

Novel State Observation Activity: Active: Technically the agent will not know the size changes continue into other rooms before visiting them. However, this may act a lot like passive.

Novelty Frequency: Less Frequent than Observation: Novel “changes” are transitions to new states, which we model as happening at the same rate as observation (in discrete time).

Hierarchy Level Analysis

-  *Analyzed level matches intended level:* An environmental fluent, “block-type-at”, is affected.
-  *Higher level novelties are not present:* While no obvious problems exist, space seems likely to have lots of follow-on effects, that could include existing non-environmental events; for example, an event that transfers objects between player and trader inventories may exist that’s dependent on them being within range of one another. (The checkmark is a slightly different color to remind you to check for possible follow-on effects.)
-  *Simpler novelty is not possible:* The transformation sequence cannot be reasonably reduced.

Target Level: 6 - Environment

Novelty Description:

- 5) Attributes: World Size
 - a. Impact: The size and/or density of objects changes
 - ~~b. More objects~~
 - ~~i. Arena gets bigger and more objects appear~~
 - c. Same # of objects
 - i. Arena gets smaller/bigger and object # stays the same

Note: Struck out text from Eric's novelty description indicates material not covered in this section.

Assumptions:

- Number of objects stays the same.
- Density is a new property.
- Density affects number of resources created by destroying an object.
- Original distribution of resources created is capped at a low integer.
- Density cannot be observed.

Estimated existing T-SAL action and event:

```
(:action break
:parameters (?block - block)
:precondition (and (is-destructible ?block) (<= (strength ?block) 1)
                  (neq (object-position ?block) NO-POSITION)
                  (eq (resource-type ?block) ?rt) (eq (available-resources ?block) ?amt)
                  (eq (object-position ?block) ?pos))
:effect (and (set (object-position ?block) NO-POSITION) (create resource-cache ?rc)
             (set (object-position ?rc) ?pos) (set (resource-count ?rc) ?amt)
             (set (resource-type ?rc) ?rt))
)
```

Estimated existing T-SAL resource amount generator:

```
RELATIONGENERATOR("available-resources", ALLPERMUTATIONS([TreeBlocks], GAUSSIANDISTRIBUTION(3,
1)))
```

Estimated T-SAL Transformations:

```
ADDERELATIONGENERATOR("available-resources", ALLPERMUTATIONS([TreeBlocks],
GAUSSIANDISTRIBUTION(6, 2)))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Increase or decrease in change size Breaks create more or fewer resources; if every resource is a separate object, this is **Added/Missing Change**

Novelty Entry Determinism: Deterministic Novel transitions are reached deterministically by destroying objects.

Novelty Obviousness: Statistical Novelty Only The distribution of resources changes, which requires the agent to compare distribution estimations.

Novelty Observation Determinism: None No change in observation probabilities.

Novelty Observation Localization: Nearby New break transitions happen before agent has chance to observe any different states.

Novel State Observation Activity: Active: Agent will not be able to differentiate density of object from original environment without destroying them.

Novelty Frequency: Less Frequent than Observation: Only happens as a result of agent actions.

Hierarchy Level Analysis



Analyzed level matches intended level: This modifies the generation of an environmental fluent.



Higher level novelties are not present: We assume that no other events are affected by the “available-resources” fluent, only actions, which is a little tricky, since what is a direct effect and an indirect effect can be arbitrary. However, it’s reasonable to assume that only the direct effects of the break action, and not non-environmental events, are affected.



Simpler novelty is not possible: The transformation sequence cannot be reasonably reduced.

Target Level: 7 - Goals

Novelty Description:

1/2) Attribute: Actors prioritize Diamond/Platinum

a. Pogoist

Assumptions:

- Pogoist goal is such that time of obtaining diamond is important
- Without that goal, POGOIST would deterministically obtain diamonds in a different order. (As dictated in task specifications)

Discussion:

I analyzed this with respect to diamond, but platinum seems a trivial change; handling pogoist case here, trader case in next analysis.

Estimated T-SAL description of original goals

```
(:metric POGOIST (if (has-pogo-stick ?ag) 100000
                    ;else
                    0
                )
)
```

Estimated T-SAL description of novel goals

```
(:metric POGOIST (+ (if (has-pogo-stick ?self) 100000
                    ;else
                    0
                )
                (* -1 (most-recent-time-obtained diamond ?ag))
            )
)
```

Estimated T-SAL Transformations:

```
REPLACEPERFORMANCEEQUATION(POGOIST,
    '(+ (if (has-pogo-stick ?self) 100000
        ;else
        0
    )
    (* -1 (most-recent-time-obtained diamond ?ag)))
)
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Pogoist acts differently.

Transition Novelty Type: New changes, Missing Changes At some point in time, pogoist must move toward diamonds rather than another resource which would be expected.

Novelty Entry Determinism: Infrequent Nondeterministic Novel transitions don't start to occur until pogoist plan/policy dictates going for diamonds next. This is deterministic to the pogoist, but as the decision is based partly on mental state internal to the pogoist, it is essentially nondeterministic to an external observing system.

Novelty Obviousness: Novel Transition Discernible Player in the same room as pogoist will see the pogoist make moves that are inconsistent with original distribution.

Novelty Observation Determinism: Deterministic With full knowledge of camera angle, occlusion, and room, the likelihood of witnessing a pogoist action on the camera is near-deterministic. With relational observations, it's just deterministic.

Novelty Observation Localization: Nearby Pogoist will take novel actions, transitioning the environment state, before the agent gets a chance to witness.

Novel State Observation Activity: Passive: Novel states will be generated primarily with respect to the inventory of the pogoist. There's likely no way for the agent to distinguish these, though.

Novelty Frequency: Less Frequent than Observation: Pogoist moves once for every two player moves.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* We can model this successfully in T-SAL as a goal change. Due to the internal reasoning process generating different plans, it seems more like an animate agent affecting the environment than an inanimate one.
- ✓ *Higher level novelties not present:* This is not also at a higher novelty level.
- ✓ *Simpler novelty not possible:* There is no simpler transformation sequence that counts as a novelty.

Target Level: 7 - Goals

Novelty Description:

- 1/2) Attribute: Actors prioritize Diamond/Platinum
b. Traders

Discussion:

I analyzed this with respect to diamond, but platinum seems a trivial change; handling trader case only here.

Assumptions:

- Traders choose trade policies dynamically to maximize an internal goal metric.

Estimated T-SAL description of original goals

```
(:metric TRADERS (+ (* 1 (resource-inventory ?self log))  
                    (* 9 (resource-inventory ?self titanium))  
                    (* 8 (resource-inventory ?self platinum))  
                    (* 1.6 (resource-inventory ?self diamond))))
```

Estimated T-SAL description of novel goals

```
(:metric TRADERS (+ (* 1 (resource-inventory ?self log))  
                    (* 9 (resource-inventory ?self titanium))  
                    (* 8 (resource-inventory ?self platinum))  
                    (* 3 (resource-inventory ?self diamond))))
```

Estimated T-SAL Transformations:

```
REPLACEPERFORMANCEEQUATION(TRADERS,  
    '(+ (* 1 (resource-inventory ?self log))  
        (* 9 (resource-inventory ?self titanium))  
        (* 8 (resource-inventory ?self platinum))  
        (* 3 (resource-inventory ?self diamond))))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Traders act differently.

Transition Novelty Type: New changes, Missing Changes Traders require more diamonds/give more in exchange for diamonds.

Novelty Entry Determinism: Deterministic Always happens with player trades with trader.

Novelty Obviousness: Novel Transition Discernible Player attempting to trade diamonds or interact will observe different trading behavior.

Novelty Observation Determinism: Deterministic New interact observations should describe new rates.

Novelty Observation Localization: Mixed Player can interact, receiving observation, or trade, possibly failing, first.

Novel State Observation Activity: Active: Exploring agent may discover ability to reach state with more resources than were available previously.

Novelty Frequency: Less Frequent than Observation: Novelty occurs when agent interacts or trades with trader only.

Hierarchy Level Analysis

✓ ***Could be modelled at lower level:*** We can model this successfully in T-SAL as a goal change. However, some learning systems might treat this as an action novelty, with the traders being essentially vending machines. It's impossible to avoid this problem completely, as even a very complex policy might be attributed to "inanimate" dynamics. However, a more dynamic policy may be more in the "spirit" of the goal novelty level. ***Possible remedy:*** If the traders change their trading policies based on scarcity and availability of resources it might seem more person-like.

✓ ***Higher level novelties not present:*** This is not also at a higher novelty level.

✓ ***Simpler novelty not possible:*** There is no simpler transformation sequence that counts as a novelty.

Target Level: 7 - Goals

Novelty Description:

3) Class: Pogoist wants all wood resources

Assumptions:

- Pogoist will use up all of the trees.

Discussion:

Previously considered a goal “(count (?o – object) (and (eq (object-type ?o) tree) (is-destroyed ?o)))” that just hates trees for existing.

Estimated T-SAL description of novel goals

```
(:metric POGOIST (inventory-count ?ag wood))
```

Estimated T-SAL Transformations:

```
REPLACEPERFORMANCEEQUATION(POGOIST,  
    '(inventory-count ?ag wood))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Pogoist acts differently.

Transition Novelty Type: New changes, Missing Changes: At some point in time, pogoist will destroy trees rather than look for the next resource.

Novelty Entry Determinism: Infrequent Nondeterministic Novel transitions don't start to occur until pogoist plan/policy dictates going for a tree instead of next resource. This is deterministic to the pogoist, but as the decision is based partly on mental state internal to the pogoist, it is essentially nondeterministic to an external observing system.

Novelty Obviousness: Novel Transition Discernible Player in the same room as pogoist will see the pogoist make moves that are inconsistent with original distribution.

Novelty Observation Determinism: Deterministic With full knowledge of camera angle, occlusion, and room, the likelihood of witnessing a pogoist action on the camera is near-deterministic. With relational observations, it's just deterministic.

Novelty Observation Localization: Nearby Pogoist will take novel actions, transitioning the environment state, before the agent gets a chance to witness.

Novel State Observation Activity: Passive: Novel states will be generated primarily with respect to the inventory of the pogoist. There's likely no way for the agent to distinguish these, though.

Novelty Frequency: Less Frequent than Observation: Pogoist moves once for every two player moves.

Hierarchy Level Analysis



✓ *Analyzed level matches intended level:* We can model this successfully in T-SAL as a goal change. Due to the internal reasoning process generating different plans, it seems more like an animate agent affecting the environment than an inanimate one.



Higher level novelties not present: This is not also at a higher novelty level.



Simpler novelty not possible: There is no simpler transformation sequence that counts as a novelty.

Target Level: 7 - Goals

Novelty Description:

- 4) Class: Traders only want logs
 - a. Traders change their trades to give either normal trades or special trades in exchange for logs/diamond/platinum.
 - b. Normal trades
 - c. Special trades (pogo stick, valuable items)

Assumptions:

- Traders choose trade policies dynamically to maximize an internal goal metric
 - See set-trade-rate action below
- Note that in the original environment, the agent probably never changed policies, so the learning system won't have already modelled the action (so effectively the action will appear to be part of the novelty, not just the goal).

Estimated T-SAL description of original goals

```
(:metric TRADERS (+ (* 1 (resource-inventory ?self log))
                    (* 9 (resource-inventory ?self titanium))
                    (* 8 (resource-inventory ?self platinum))
                    (* 1.6 (resource-inventory ?self diamond))))
```

Estimated T-SAL description of original actions

```
(:action set-trade-rate
:parameters (?rtIn – resource-type ?rtOut – resource-type ?amtIn – integer ?amtOut – integer)
:performer ?trader
:precondition ()
:effect (offers-trade ?trader ?rtIn ?rtOut ?amtIn ?amtOut)
)
```

Estimated T-SAL description of novel goals

```
(:metric TRADERS (+ (* 1 (resource-inventory ?self log))
```

Estimated T-SAL Transformations:

```
REPLACEPERFORMANCEEQUATION(TRADERS,
                              '(+ (* 1 (resource-inventory ?self log))))
```

Discussion:

Note that in the original environment, a learning system probably never had reason to postulate the existence of a “set-trade-rate” action like the one above, so it may appear to be novel even though it’s not.

These ones that assume that traders trading policies has always been goal-based but suddenly change makes me think of a philosophical paradox – people who do the same thing time after time are likely to enrage people when they change their behavior. Unpredictable, chaotic types don’t enrage the same way because people don’t have strong expectations about them. But the unpredictable people were less reliable to begin with. Maybe it’s because we used to be able to predict these people with a simple model, and now that they aren’t simple, we have to do more work to be able to predict them again.

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Traders act differently.

Transition Novelty Type: New changes, Missing Changes Traders give different things in exchange for logs.

Novelty Entry Determinism: Deterministic Always happens with player trades with trader.

Novelty Obviousness: Novel Transition Discernible Player attempting to trade diamonds or interact will observe different trading behavior.

Novelty Observation Determinism: Deterministic New interact observations should describe new rates.

Novelty Observation Localization: Mixed Player can interact, receiving observation, or trade, possibly failing, first.

Novel State Observation Activity: Active: Exploring agent may discover ability to reach state with more resources than were available previously.

Novelty Frequency: Less Frequent than Observation: Novelty occurs when agent interacts or trades with trader only.

Hierarchy Level Analysis

✓ ***Could be modelled at lower level:*** We can model this successfully in T-SAL as a goal change. However, some learning systems might treat this as an action novelty, with the traders being essentially vending machines. It’s impossible to avoid this problem completely, as even a very complex policy might be attributed to “inanimate” dynamics. However, a more dynamic policy may be more in the “spirit” of the goal novelty level. ***Possible remedy:*** If the traders changed their policies to want whatever is least available (like collectors?) it might be more human-like.

✓ ***Higher level novelties not present:*** This is not also at a higher novelty level.



Simpler novelty not possible: There is no simpler transformation sequence that counts as a novelty.

Target Level: 7 - Goals

Novelty Description:

- 5) Class: Traders only want atypical items
 - a. Traders offer trades for items like the pickaxe, tree tap, blue key, or a crafting table

Assumptions:

- Traders choose trade policies dynamically to maximize an internal goal metric
 - See set-trade-rate action below
- Note that in the original environment, the agent probably never changed policies, so the learning system won't have already modelled the action (so effectively the action will appear to be part of the novelty, not just the goal).

Estimated T-SAL description of original goals

(:metric TRADERS (+ (* 1 (resource-inventory ?self log))
(* 9 (resource-inventory ?self titanium))
(* 8 (resource-inventory ?self platinum))
(* 1.6 (resource-inventory ?self diamond))))

Estimated T-SAL description of novel goals

(:metric TRADERS (+ (* 1 (resource-inventory ?self pickaxe)))

Estimated T-SAL Transformations:

REPLACEPERFORMANCEEQUATION(TRADERS,
'(+ (* 1 (resource-inventory ?self pickaxe))))

Discussion:

While this is different from an accommodation standpoint compared to the logs one, it's not really different from a detection and characterization standpoint.

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Traders act differently.

Transition Novelty Type: New changes, Missing Changes Traders give different things in exchange for logs.

Novelty Entry Determinism: Deterministic Always happens with player trades with trader.

Novelty Obviousness: Novel Transition Discernible Player attempting to trade or interact will observe different trading behavior.

Novelty Observation Determinism: Deterministic New interact observations should describe new rates.

Novelty Observation Localization: Mixed Player can interact, receiving observation, or trade, possibly failing, first.

Novel State Observation Activity: Active: Exploring agent may discover ability to reach state with more resources than were available previously.

Novelty Frequency: Less Frequent than Observation: Novelty occurs when agent interacts or trades with trader only.

Hierarchy Level Analysis

✓ ***Could be modelled at lower level:*** We can model this successfully in T-SAL as a goal change. However, some learning systems might treat this as an action novelty, with the traders being essentially vending machines. It's impossible to avoid this problem completely, as even a very complex policy might be attributed to "inanimate" dynamics. However, a more dynamic policy may be more in the "spirit" of the goal novelty level. ***Possible remedy:*** If the traders offered services such as help chopping or raised prices or had goals to save trees (Lorax?) such that they'd refuse to trade if you'd chopped one down, the changes might be more pronounced and animate.

✓ ***Higher level novelties not present:*** This is not also at a higher novelty level.

✓ ***Simpler novelty not possible:*** There is no simpler transformation sequence that counts as a novelty.

Target Level: 8 - Events

Novelty Description:

1/2) Class: Resources/Agents come and go

Assumptions:

Resources and/or continue to exist when they disappear, but are not seen on camera or in relational description.

All objects disappear or reappear at random times; traders, pogoists, trees and ore are meaningful subsets but that doesn't change analysis.

Disappearances and reappearances happen on a schedule with timing the same as that of actions and observations.

Discussion:

Probably only one of "disappears" and "reappears" is necessary per novelty. This changes observation function, which we don't deal with much, and technically the current version of T-SAL doesn't handle observations, so this is an extrapolation. Below estimated observation event just says that now an object's type and position are observed *only when visible*.

Estimated modified T-SAL observation event:

```
(:event object-observed
:observation t
:parameters (?o - object)
:performer ?ag
:condition (and (object-visible ?o))
:observe (and (eq (object-type ?o) ?t) (eq (object-type ?o) ?pos))
)
```

Note: Green text indicates new preconditions and effects.

Estimated new T-SAL events:

```
(:event object-disappears
:parameters (?o - object)
:precondition (and (eq (clock-time) ?tm) (object-phase-time ?o ?tm) (object-visible ?o))
:effect (and (not (object-visible ?o)))
)
```

```
(:event object-appears
:parameters (?o - object)
```

```
:precondition (and (eq (clock-time) ?tm) (object-phase-time ?o ?tm) (not (object-visible ?o)))
:effect (and (not (object-visible ?o)))
)
```

Estimated T-SAL Transformations:

```
ADDFLUENT("object-visible")
ADDFLUENT("object-phase-time")
ADDEVENT(object-disappears)
ADDPRECONDITION(object-disappears, "(eq (clock-time) ?tm)")
ADDPRECONDITION(object-disappears, "(object-phase-time ?o ?tm)")
ADDPRECONDITION(object-disappears, "(object-visible ?o)")
ADDEFFECT (object-disappears, "(not (object-visible ?o))")
ADDEVENT(object-appears)
ADDPRECONDITION(object-appears, "(eq (clock-time) ?tm)")
ADDPRECONDITION(object-appears, "(object-phase-time ?o ?tm)")
ADDPRECONDITION(object-appears, "(not (object-visible ?o))")
ADDEFFECT (object-appears, "(object-visible ?o)")
ADDRELATIONGENERATOR("object-visible", ALLPERMUTATIONS([AllBlocks], CONSTANTFUNCTION(TRUE)))
ADDRELATIONGENERATOR("object-phase-time", NRELATIONDRAWS([AllBlocksWithReplacement,
UNIFORMDISTRIBUTION(0, 10000)], CONSTANTFUNCTION(TRUE)), 500) )
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: None All transitions should still be possible; only new properties and observations differ.

Novelty Entry Determinism: Infrequent Nondeterministic Novel observations happen at random times determined by the scenario.

Novelty Obviousness: Alien Observation Blocks that are holding things up might disappear, making things appear to hover in midair.

Novelty Observation Determinism: Frequent Nondeterministic – In general, objects could be disappeared or not with a 50% likelihood based on current estimates. The frequency is assumed.

Novelty Observation Localization: Distant: Observations are modified, but transitions are preserved.

Novel State Observation Activity: Active: Agent shouldn't be able to tell the difference between a disappeared and missing block without attempting to interact with it.

Novelty Frequency: Less Frequent than Observation: Assume essentially discrete time, such that this happens no more frequently than actions and observations.

Hierarchy Level Analysis

 *Analyzed level **does not match** intended level:* New environmental fluents “object-visible” and “object-phase-time” and environmental events (“object-appears” and object-disappears”) are created. All objects are affected uniformly. These seem to fall squarely within our definition of the environment level. **Possible Remedy:** In our definition, for the event to be non-environmental, the agent must have some interaction with it. Thus, if only things the agent touched or walked near had a probability of a disappearing, it would be an event novelty.

 *Higher level novelties not present:* Highest novelty level.

 *Simpler novelty **is possible**:* Removals from the transformation sequence are compatible with the same novelty level. **Possible Remedy:** At the environment level, simply removing object-appears would resolve this problem.

Target Level: 8 - Events

Novelty Description:

3) Class: Earthquake

Assumptions:

Assume earthquake affects entire area simultaneously

Assume earthquake moves objects randomly with gaussian distribution

Assume “disappear” means “goes away entirely”

Earthquake is not observed directly.

Discussion:

This is another that our formalization places as environment rather than event; I’d like to discuss how you define this boundary.

Estimated new T-SAL events:

```
(:event earthquake-destroys
:parameters (?o - object)
:precondition (and (eq (clock-time) ?tm) (earthquake-time ?tm))
:effect (and (set (object-position ?o) NO-POSITION))
)
```

```
(:event earthquake-moves
:parameters (?o - object)
:precondition (and (eq (clock-time) ?tm) (earthquake-time ?tm) (eq (object-position ?o) ?pos)
:effect (and (set (object-position ?o) (:gaussian2dim ?pos 5)) )
)
```

Estimated T-SAL Transformations:

```
ADDFLUENT(“earthquake-time”)
ADDEVENT(earthquake-destroys)
ADDPRECONDITION(earthquake-destroys, “(eq (clock-time) ?tm)”)
ADDPRECONDITION(earthquake-destroys, “(earthquake-time ?tm)”)
ADDEFFECT (earthquake-destroys, “(set (object-position ?o) NO-POSITION)”)
ADDEVENT(earthquake-moves)
ADDPRECONDITION(earthquake-moves, “(eq (clock-time) ?tm)”)
ADDPRECONDITION(earthquake-moves, “(earthquake-time ?tm)”)
```

```
ADDPRECONDITION(earthquake-moves, "(eq (object-position ?o) ?pos)")
ADDEFFECT (earthquake-moves, "(set (object-position ?o) (:gaussian2dim ?pos 5))")
ADDRATIONGENERATOR("earthquake-time", NRELATIONDRAWS([UNIFORMDISTRIBUTION(0, 10000)],
CONSTANTFUNCTION(TRUE), 1) )
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Added Changes Objects will move and disappear at times that wouldn't have before.

Novelty Entry Determinism: Infrequent Nondeterministic The earthquake time is determined by the scenario.

Novelty Obviousness: Novel Transition Discernible The effects of the earthquake should be immediately noticeable after it occurs.

Novelty Observation Determinism: None Individual observations are unaffected.

Novelty Observation Localization: Nearby No warning given.

Novel State Observation Activity: Passive: No novel states, just transitions.

Novelty Frequency: Less Frequent than Observation: Assume essentially discrete time, such that this happens no more frequently than actions and observations.

Hierarchy Level Analysis

 **Analyzed level *does not match* intended level:** New environmental fluents "earthquake-time" and environmental events ("earthquake-destroys" and "earthquake-moves") are created. All objects are affected uniformly. These seem to fall squarely within our definition of the environment level. **Possible Remedy:** In our definition, for the event to be non-environmental, the agent must have some interaction with it. Thus, if the earthquake was a landslide that happened after the agent picked away at ore too much, it would be an event novelty.

 **Higher level novelties not present:** Highest novelty level.

 **Simpler novelty *is possible*:** Removals from the transformation sequence are compatible with the same novelty level. **Possible Remedy:** At the environment level, simply removing earthquake-destroys or earthquake-moves would resolve this problem.

Target Level: 8 - Events

Novelty Description:

4) Class: Items appear

Assumptions:

Items come into existence, and can only be interacted with afterward.

Objects appear at random times.

Discussion:

This is another that our formalization places as environment rather than event; I'd like to discuss how you define this boundary.

Estimated new T-SAL events:

```
(:event item-appears
:parameters (?o - object)
:precondition (and (eq (clock-time) ?tm) (item-appearance ?tm ?pos ?o))
:effect (and (set (object-position ?o) ?pos))
)
```

Estimated T-SAL Transformations:

```
ADDFLUENT("item-appearance")
ADDEVENT(item-appears)
ADDPRECONDITION(item-appears, "(eq (clock-time) ?tm)")
ADDPRECONDITION(item-appears, "(item-appearance ?tm ?pos ?o)")
ADDEFFECT (item-appears, "(set (object-position ?o) ?pos)")
ADDRATIONGENERATOR("item-appearance", NRELATIONDRAWS([UNIFORMDISTRIBUTION(0, 10000),
AllPositions, NDRWS(AllObjects, 50)), CONSTANTFUNCTION(TRUE), 50) )
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: Added Changes Objects will appear out of nowhere, unpredictably.

Novelty Entry Determinism: Infrequent Nondeterministic Nonpredictable, infrequent item appearances.

Novelty Obviousness: Novel Transition Discernible If present, agent should see item appear.

Novelty Observation Determinism: None Individual observations are unaffected.

Novelty Observation Localization: Nearby No warning given.

Novel State Observation Activity: Passive: No novel states, just transitions.

Novelty Frequency: Less Frequent than Observation: Assume essentially discrete time, such that this happens no more frequently than actions and observations.

Hierarchy Level Analysis

 *Analyzed level **does not match** intended level:* New environmental fluent “item-appearance” and environmental event (“item-appears”) are created. These seem to fall squarely within our definition of the environment level. **Possible Remedy:** In our definition, for the event to be non-environmental, the agent must have some interaction with it. Thus, if items appeared in front of the agent, or only right after the player chopped down a tree, for example, it would be an event novelty.

 *Higher level novelties not present:* Highest novelty level.

 *Simpler novelty is not possible:* Removing items from the transformation sequence does not result in a sensible novelty.

Target Level: 8 - Events

Novelty Description:

5) Class: Wildfire

- a. Fire spawns and despawns and the agent dies if they touch it.
- b. Fire spawns and despawns and the agent has a massive negative stepCost if they touch it.

Assumptions:

Fire did not exist in source domain.

Fire can be observed from a distance.

Fire spawns and despawns happen on same discrete schedule as agent actions.

Discussion:

This is another that our formalization places as environment rather than event; I'd like to discuss how you define this boundary. Note that there is a difference between this fire and the other fire example in that this fire burns when you "touch" it not "move onto" it. I assume that this fire keeps burning, and therefore appears less similar to an effect of the movement action.

Estimated new T-SAL events:

```
(:event fire-spawns
:parameters (?pos - position)
:precondition (and (eq (clock-time) ?tm) (fire-spawn-time ?pos ?tm))
:effect (and (fire-at ?pos))
)
```

```
(:event fire-despawns
:parameters (?pos - position)
:precondition (and (eq (clock-time) ?tm) (fire-despawn-time ?pos ?tm))
:effect (and (not (fire-at ?pos)))
)
```

```
(:event fire-burns
:parameters (?ag - agent)
:precondition (and (eq (object-position ?ag) ?pos) (fire-at ?pos))
:effect (and (increase (cost) 10000))
)
```

Estimated T-SAL Transformations:

```

ADDFLUENT("fire-spawn-time")
ADDFLUENT("fire-despawn-time")
ADDFLUENT("fire-at")
ADDEVENT(fire-spawns)
ADDPRECONDITION(fire-spawns, "(eq (clock-time) ?tm)")
ADDPRECONDITION(fire-spawns, "(fire-spawn-time ?pos ?tm)")
ADDEFFECT (fire-spawns, "(fire-at ?pos)")
ADDEVENT(fire-despawns)
ADDPRECONDITION(fire-despawns, "(eq (clock-time) ?tm)")
ADDPRECONDITION(fire-despawns, "(fire-despawn-time ?pos ?tm)")
ADDEFFECT (fire-despawns, "(not (fire-at ?pos))")
ADDEVENT(fire-burns)
ADDPRECONDITION(fire-burns, "(eq (object-position ?ag) ?pos)")
ADDPRECONDITION(fire-burns, "(fire-at ?pos)")
ADDEFFECT (fire-burns, "(increase (cost) 10000)")
ADDVALUEGENERATOR(FireSpawnLocations, NDRAWS(AllPositions, 50))
ADDVALUEGENERATOR(SpawnDespawnTimes,
  FILTER(
    DRAWTUPLE([UNIFORMDISTRIBUTION(0, 10000), UNIFORMDISTRIBUTION(0, 10000)],
      ["earlier", "later"]),
    <))
ADDRELATIONGENERATOR("fire-spawn-time", ALLPERMUTATIONS([FireSpawnLocations],
SpawnDespawnTimes.earlier) )
ADDRELATIONGENERATOR("fire-despawn-time", ALLPERMUTATIONS([FireSpawnLocations],
SpawnDespawnTimes.later) )

```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: New Changes Fire spawns, despawns, burns. All new.

Novelty Entry Determinism: Infrequent Nondeterministic Fire pops up at random times determined by scenario.

Novelty Obviousness: Alien Observation Fire is a new type of observation.

Novelty Observation Determinism: Deterministic – Mostly deterministic, with possible caveats about cameras.

Novelty Observation Localization: Mixed: Usually agent will observe the fire before it interacts with the state meaningfully, but it might spawn on top of the agent.

Novel State Observation Activity: Active: Fire could spawn in another room, and agent wouldn't know until it moves.

Novelty Frequency: Less Frequent than Observation: Assume essentially discrete time, such that this happens no more frequently than actions and observations.

Hierarchy Level Analysis

✓ *Analyzed level matches intended level:* The “fire-burns” event is not an environmental event as it depends on the agent’s position. Therefore, we read this as event-level novelty.

✓ *Higher level novelties not present:* Highest novelty level.

✗ *Simpler novelty **is possible**:* There are three events introduced; only one is needed.
Possible Remedy: If you remove the fire spawn and despawn, and just have the agent interact with the fire burning, this is a simpler novelty. You might want to ensure that some interaction with fire is required (via blocking bottlenecks) so that fire isn’t just an obstacle to route around, but to minimize contact with.

If that seems too boring, removing just the despawn would still be in the spirit of decomplicating.

Novelty Analysis for Washington State University Novelty Generation Team

Matthew Molineaux – Parallax Advanced Research – matthew.molineaux@parallaxresearch.org
Prepared with support from PacMar Technologies

August 19, 2022



SBIR Data Rights

Contract No. HR001121C0236

Contractor Name : PacMar Technologies

Address: 841 Bishop Street, Ste. 1110, Honolulu, HI 96813

Expiration of SBIR Data Rights Period : 17 Sept 2041

The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software - Small Business Innovation Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.

Analysis Purpose and Method

We are providing feedback to help your decision-making process on the SAIL-ON program as a TA2, not attempting to call your decisions into question. Our primary goal is to provide you with descriptions of the novelty dimensions and estimated difficulties of them, for you to consider; and to provide you with our assessments of how well they meet the novelty hierarchy level as we have formalized it in our theory (see Transformation and Hierarchy Formalization document).

For each novelty below, we describe the following elements:

- Target Level: The novelty hierarchy level you targeted
- Novelty Description: Your description of the novelty
- Assumptions: Assumptions we made about the novelty that may or may not be correct.
- Discussion: We use this section to provide context for how we look at the novelty and how we model it.
- Estimated T-SAL Environment Model Elements: Excerpts from a hypothetical T-SAL model of the modified and/or original environment that describe the novelty.
- Estimated T-SAL Transformation Sequence: Formal description of the novelty as a sequence of transformations between initial environment model and scenario generator to modified environment model and scenario generator; as described in the Transformation and Hierarchy Formalization document
- Novelty Dimensions: Assignment of novelty characteristics along eight environment transformation dimensions described in our Environment Transformation Framework theory, and explanation of each assignment
- Hierarchy Level Analysis: An analysis of whether the described novelty fits the definitions of hierarchy levels in the SAIL-ON program, based on our formal theory describing each.

Our assessments of novelty dimension values and hierarchy level analysis are based on the T-SAL language (itself primarily based of PDDL+) and transformations to environments described in that language. As such, the estimated transformation sequence is the formal basis for our determinations, and it's based on the estimated environment model elements required to describe a novelty. In many cases, multiple transformation sequences could create different environment models that would explain the observations generated equally well. When we see this as being possible, we have noted it.

The hierarchy level analysis for each novelty answers three questions:

1. Does the novelty, as described, clearly have the characteristics of the stated level as described in our theory? Might it be described at a lower one?
2. Does the novelty **also** have characteristics from a higher novelty level, which might make it more difficult than intended?

3. Is the novelty as simple as possible? This is related to PM Ted Senator’s request that only a “single” novelty be presented at each point. Our transformation sequences make clear that it is usually not possible to describe a novelty with only a single textual change to a model, so we ask instead whether a simpler transformation sequence can be constructed, by removing one or more existing elements, that still meets the requirements of the target novelty level. There may be circumstances in which meeting this criterion is impractical.

These questions have the following possible responses:

Q	Icon	Response	Meaning
1		<i>Analyzed level matches intended level</i>	Our theory of hierarchy levels treats the level of the novelty as the same one you provided.
		<i>Could be modelled at lower level</i>	We found a model for the novelty as described that matches the provided level in our theory of hierarchy levels, but we also found an alternative model at a lower novelty level. This might confuse some agents.
		<i>Analyzed level does not match intended level</i>	We did not find a model for the novelty that matches your provided level.
2		<i>Higher level novelties not present</i>	The transformation sequence found does not also meet the conditions for a higher level novelty than that stated.
		<i>Higher level novelties are present</i>	The transformation sequence found meets the conditions for a higher level novelty than that stated, in addition to conditions for a lower level.
3		<i>Simpler novelty not possible</i>	There is no way to remove a transformation from the T-SAL transformation sequence that models this novelty and arrive at a sequence from the described level.
		<i>Simpler novelty is possible</i>	One or more T-SAL transformations could be removed from this sequence, and the resulting sequence would still be a legal novelty at the intended level. (That novelty might be a nuisance novelty, though.)

In addition to the icon and response, we describe the rationale for each response, and provide a possible remedy, if you are interested in pursuing it, for “non-green” responses. There’s nothing binding about these, we’re just trying to provide the best feedback we can; please ignore whatever’s not helpful to you.

Cart-Pole

Target Level: 6 - Environment

Novelty Description: Wind force on all objects. E/M/H based on amount of wind.

Assumptions:

- Continuous time physics
- Wind exerts same force on all objects (i.e., constant in space)
- The pole is affected differently by the wind, since it is fixed in position at one end, resulting in angular acceleration
- Wind force is constant. Assume wind is NOT observed directly.

Estimated T-SAL description of novel event

```
(:process wind-moves-block
:parameters (?block – object)
:precondition ()
:effect (and (increase (block-acc ?block) (/ #t (wind-force) (block-mass ?block) )))
)
```

```
(:process wind-moves-cart
:parameters (?cart – object)
:precondition ()
:effect (and (increase (cart-acc ?cart) (/ #t (wind-force) (cart-mass ?cart))))
)
```

```
(:process wind-moves-pole
:parameters (?cart – object)
:precondition ()
:effect (and (increase (pole-angle-acc ?cart) (convert-angle (/ #t (wind-force) (pole-mass ?cart)))))
)
```

Estimated T-SAL Transformation Sequence:

```
ADDFLUENT(wind-force) //Not observed
ADDPROCESS(wind-moves-block)
ADDPROCESSCHANGE(wind-moves-block, "(increase (block-acc ?block) ...)")
ADDPROCESS(wind-moves-cart)
ADDPROCESSCHANGE(wind-moves-cart, "(increase (cart-acc ?cart) ...)")
ADDPROCESS(wind-moves-pole)
ADDPROCESSCHANGE(wind-moves-pole, "(increase (pole-angle-acc ?cart) ...)")
```

ADDERELATIONGENERATOR("wind-force", UNIFORMDISTRIBUTION(0/EasyMax/MediumMax, EasyMax/MediumMax/HardMax))

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: Physics changes only.

Transition Novelty Type: New changes, Increase/Decrease in Change Value Wind added, so objects will accelerate and move that would have been stationary or constant velocity in original environment. Also, velocities and positions of objects moving in wind direction will sometimes increase more than expected.

Novelty Entry Determinism: None Entry is immediate, since wind is constant over time.

Novelty Obviousness: Novel Transition Discernible Different changes in position should be happening constantly, and observed immediately. (**Alien Observation** if wind is observed directly.)

Novelty Observation Determinism: None Assuming the wind is not observed directly, so there's no change.

Novelty Observation Localization: Nearby Agent always encounters novel transition first.

Novel State Observation Activity: Passive: All states are distinguishable, so not needed.

Novelty Frequency: Continuous: Treating wind as a continuous effect, although since the speed does not change with time, it could be modelled as a discrete time effect with a period equivalent to the observational period without loss of information.

Hierarchy Level Analysis



Analyzed level matches intended level: This is an environmental novelty, as it modifies an environmental fluent (wind-force).



Higher level novelties not present: This is not also at a higher novelty level: no situational events are added or modified, ruling out event novelty; the task performance definition is unmodified, ruling out goal novelty



Simpler novelty is possible: Adding a single process is necessary so that wind-force has an effect on the environment. However, a simpler environment novelty sequence can be constructed by removing some process-based changes. **Possible remedy:** By ensuring the wind does not affect the pole angle, only positions of objects, a process that moves only blocks and cart in an identical fashion would be simpler.

Target Level: 7 - Goals

Novelty Description: Blocks constantly attracted to pole. E/M/H based on amount by which blocks attracted to pole.

Assumptions:

- Speed of blocks is constant
- Direction of block movement changes once per action cycle.
- Action cycle for block movement is same as cart movement.
- Block action changes velocity to arbitrary vector.
- Easy/medium/hard differ only in the policy distribution of the agents, since blocks could choose any velocity.

Discussion:

Under the blocks-as-agents paradigm, the blocks are agents which choose what direction to move in. If those blocks have full control over their direction of movement at any time, giving them a goal of touching the pole would give the appearance of them simply being attracted to the pole. The introduction of a magnetic field, which involves no volition at all, might cause identical behavior and exist at a lower novelty level.

Estimated T-SAL description of novel goals

(:metric BLOCK (- (distance (block-center ?block) (pole-axis ?cart))))

Estimated T-SAL Transformations:

REPLACEPERFORMANCEEQUATION(BLOCK, '(- (distance (block-center ?block) (pole-axis ?cart))))

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Blocks act differently.

Transition Novelty Type: New changes Block velocities change over time.

Novelty Entry Determinism: Deterministic Blocks change velocity no matter what the agent does, without noise.

Novelty Obviousness: Entry Transition Discernible Changes in velocity should be observed immediately when other agents react to movement.

Novelty Observation Determinism: None Nothing novel in the observation function.

Novelty Observation Localization: Nearby All observations were possible in original environment.

Novel State Observation Activity: Passive: Even if the agent doesn't move, blocks should track the pole.

Novelty Frequency: Less Frequent than Observation: Assuming since blocks are agents, they take action as often as the main agent (i.e., cart controller) does.

Hierarchy Level Analysis

- ✓ ***Could be modelled at lower level:*** We model this in T-SAL as a goal change, making it a goal novelty that block agents pursue by making velocity changes. However, some learning systems might treat this as an environment novelty, with the blocks modelled as inanimate state elements instead. It's impossible to avoid this problem completely, as even a very complex policy might be attributed to "inanimate" dynamics. However, a more complex policy may be more in the "spirit" of the goal novelty level. ***Possible remedy:*** If the blocks' achieved their goal by anticipating the cart's action, based on the current pole lean, instead of attracting to the current spot, that might appear more like a goal novelty and less like an environment novelty.
- ✓ ***Higher level novelties not present:*** This is not also at a higher novelty level: attraction of objects, if used as a basis for this novelty, would be global rather than subject to specific conditions.
- ✓ ***Simpler novelty not possible:*** There is no simpler version of the transformation sequence that is itself a novelty.

Target Level: 8 - Events

Novelty Description: New blocks spawn over time following level-0 behavior. E/M/H based on spawn rate.

Assumptions:

- Spawn rate is a constant drawn during scenario generation from a fixed interval.
- Spawned blocks take an immediate action to change velocity to an arbitrary vector.

Estimated T-SAL description of novel event

```
(:event block-spawn
:parameters ()
:precondition (≥ (- (current-time) (last-spawn)) (spawn-rate))
:effect (and (create (block) ?b)
            (set (block-position ?b) (:random-draw 3))
            (set (last-spawn) (current-time))
          )
)
```

Estimated T-SAL Transformation sequence:

```
ADDFLUENT(last-spawn)
ADDFLUENT(spawn-rate)
ADDEVENT(block-spawn)
ADDEVENTPRECONDITION(block-spawn, '(≥ (- (current-time) (last-spawn) (spawn-rate)))
ADDEVENTEFFECT(block-spawn, '(create (block) ?b))
ADDEVENTEFFECT(block-spawn, '(set (block-position ?b) (:random-draw 3))
ADDEVENTEFFECT(block-spawn, '(set (last-spawn) (current-time)))
ADDRELATIONGENERATOR("spawn-rate", UNIFORMDISTRIBUTION(0/EasyMax/MediumMax,
EasyMax/MediumMax/HardMax))
ADDDEFAULTVALUE("last-spawn", 0))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Changing numbers of agents seems to count, but this is an edge case which may need further consideration.

Transition Novelty Type: New changes Blocks appearing is a qualitatively new change.

Novelty Entry Determinism: Deterministic Block appearances happen at the spawn rate.

Novelty Obviousness: Alien Observation While all individual partial observations are non-novel, the length of the overall observation is alien.

Novelty Observation Determinism: Deterministic Velocity changes are always observed.

Novelty Observation Localization: Nearby Agent always encounters novel transition first.

Novel State Observation Activity: Passive: Even if the agent doesn't move, new blocks appear.

Novelty Frequency: Less/More Frequent than Observation? Assuming at least easy has no more than 1 new block per observation; hard may well be more frequent.

Hierarchy Level Analysis

 *Analyzed level **does not match** intended level:* This novelty belongs to the environment level according to our theory, as all fluents that the new event is conditioned on are global. The entire environment is eligible to be impacted by events. **Possible remedy:** An adjustment that caused blocks only to spawn at particular spawn locations, or in relation to other objects, would cause this to be event novelty according to our definition.

 *Higher level novelties not present:* This is not also at a higher novelty level: no higher level is defined.

 *Simpler novelty not possible:* It is not possible to remove a transformation from the sequence without rendering the novelty illegal or non-periodic (and therefore a lower novelty level).

VizDoom

Target Level: 6 - Environment

Novelty Description: Health declines over time, even if not hit. E/M/H based on rate of decline.

Assumptions:

- Health decline rate is constant
- Health decline is continuous with respect to time.

Discussion:

Health decline could instead be modelled as a discrete time event, without impact on the novelty level.

Estimated T-SAL description of novel process:

```
(:process health-degradation
:parameters (?c - character)
:precondition ()
:effect (and (decrease (character-health ?c) (* #t (health-decline-rate) ) )
)
```

Estimated T-SAL Transformation Sequence:

```
ADDFLUENT(health-decline-rate)
ADDPRECESS(health-degradation)
ADDPRECESSCHANGE(health-degradation, '(decrease (character-health ?c) (* #t (health-decline-rate) ))
ADDRELATIONGENERATOR("health-decline-rate", UNIFORMDISTRIBUTION(0/EasyMax/MediumMax, EasyMax/MediumMax/HardMax))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present No change to agent policies

Transition Novelty Type: New Change, Increase or Decrease in Change Value Hit point changes will happen in new states, or increase after transitions when agent is hit.

Novelty Entry Determinism: None Agent never inhabits state where health decline is not happening.

Novelty Obviousness: Novel Transition Discernible Every turn should see some amount of health decrease that would not happen in original environment.

Novelty Observation Determinism: None All health levels could happen in original environment.

Novelty Observation Localization: Nearby No new observations, so transitions must occur first.

Novel State Observation Activity: Passive All states are distinguishable, so no special observation needed.

Novelty Frequency: Continuous: Health decreases with time

Hierarchy Level Analysis

✓ *Analyzed level matches intended level:* This is strongly within environment novelty, as it's modelled easily with 0 preconditions, and an environmental fluent (health-decline-rate) is introduced.

✓ *Higher level novelties not present:* Goal novelty does not apply as no goals are introduced. This cannot be modelled as event novelty because the state changes introduced are not conditioned on anything.

✓ *Simpler novelty not possible:* There is no legal simpler version of this transformation sequence that is also a novelty.

Target Level: 7 - Goals

Novelty Description: Enemies move toward certain location, and if any of them get there, the player dies. E/M/H based on number of enemies with this behavior.

Assumptions:

- Player death location is constant, drawn at random for each scenario.
- Player death is caused by an action performed by the enemy who reaches the location

Discussion:

We refer to the player dies location as a “flag” position, as in a capture the flag game. If the player died due to environment state rather than an action, this would count as an event level novelty.

Estimated T-SAL description of novel action:

```
(:action raise-flag
:parameters ()
:precondition (and (is-player ?p) (= (flag-position) (character-position (self))))
:effect (and (character-dead ?p) (raised-flag))
)
```

Estimated T-SAL description of novel goal:

```
(:metric FLAG-SEEKER (raised-flag))
```

Estimated T-SAL Transformation Sequence:

```
ADDACTION(raise-flag)
ADDPRECONDITION(raise-flag, '(is-player ?p))
ADDPRECONDITION(raise-flag, '(= (flag-position) (character-position (self))))
ADDEFFECT(raise-flag, '(character-dead ?p))
ADDEFFECT(raise-flag, '(raised-flag))
ADDFLUENT(flag-position)
ADDVALUEGENERATOR(X, UNIFORMDISTRIBUTION(0.0, 100.0))
ADDVALUEGENERATOR(Y, UNIFORMDISTRIBUTION(0.0, 100.0))
ADDRELATIONGENERATOR("flag-position", NRELATIONDRAWS([X, Y], True, 1))
REPLACEPERFORMANCEEQUATION(FLAG-SEEKER, '(raised-flag))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Present Agent policies must change.

Transition Novelty Type: New Change, Modified Change Probability Depends on determinism of agent policies. Agents will sometimes be motivated to go in different directions, or not to bother shooting. If original agent was nondeterministic, these may have been within probability space of original policy.

Novelty Entry Determinism: ? Depends on agent determinism again. If new agent is

deterministic, divergence between original and modified policy will happen at same time.

Novelty Obviousness: Novel Transition Discernible A transition where an agent goes from maximum health to dead because an enemy reaches its goal will be easily distinguished from the original environment.

Novelty Observation Determinism: Deterministic When agent reaches goal, you will observe a dead player with health remaining

Novelty Observation Localization: Nearby New transitions will occur before player receives any special observations.

Novel State Observation Activity: Passive If the agent does nothing, will eventually receive strange death observation.

Novelty Frequency: Less Frequent than Observation: Enemies move at roughly same rate as player observation.

Hierarchy Level Analysis

✓ ***Could be modelled at lower level:*** This is stated as goal novelty, but if the original enemy goal were modelled as “(and (is-player ?p) (character-dead ?p))”, the new win condition can be expressed as the same goal, just a different policy for achieving it; therefore, some learning systems might treat this as an action novelty or event novelty (depending on how they can model the new player death). ***Possible remedy:*** This ambiguity would go away if the player did not die when the flag was obtained.

✓ ***Higher level novelties not present:*** Under this formulation, no higher levels apply, as no new events are added. Note that while there’s no reason to assume the death is due to an event rather than an action, it would be possible to model it that way; we do not suggest a change, as the choice to regard it as higher level is arbitrary.

✗ ***Simpler novelty is possible:*** The player death is not required for the goal novelty. ***Possible remedy:*** You could remove the player death from the novelty, and the location-seeking behavior would still be novel, although less relevant to the player’s goal. The player might still be able to use this information to optimize performance by recognizing that those agents can be ignored or anticipated better.

Target Level: 8 - Events

Novelty Description: Dead enemies respawn after delay. E/M/H based on delay.

Assumptions:

- Each enemy has a single spawn location that does not change.
- Respawn delay is a constant.

Estimated T-SAL description of novel event

```
(:event respawn
:parameters (?enemy - enemy)
:precondition (and (character-dead ?enemy)
                  (≥ (- (current-time) (enemy-death-time ?enemy)) (respawn-delay))
                )
:effect (and (set (enemy-position ?enemy) (enemy-spawn-point ?enemy))
            (not (character-dead ?enemy)) )
)
```

Estimated T-SAL Transformation sequence:

```
ADDFLUENT(FLUENT(NAME: respawn-delay ARGUMENTS: [] VALUE-TYPE: duration)
ADDEVENT(respawn)
ADDEVENTPRECONDITION(respawn, '(character-dead ?enemy))
ADDEVENTPRECONDITION(respawn, '(≥ (- (current-time) (enemy-death-time ?enemy)) (respawn-delay)) )
ADDEVENTEFFECT(respawn, '(set (enemy-position ?enemy) (enemy-spawn-point ?enemy)) )
ADDEVENTEFFECT(respawn, '(not (character-dead ?enemy)) )
ADDRELATIONGENERATOR("respawn-delay", UNIFORMDISTRIBUTION(0/EasyMax/MediumMax,
EasyMax/MediumMax/HardMax))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: A bit fuzzy, but the policies of the agents will presumably be the same after respawn.

Transition Novelty Type: New changes Enemies respawning is a qualitatively new change.

Novelty Entry Determinism: Deterministic Enemies respawn non-randomly.

Novelty Obviousness: Entry Transition Discernible As enemy positions are always known, the agent can identify that an enemy goes from dead to not-dead immediately.

Novelty Observation Determinism: None All observations are possible within original environment.

Novelty Observation Localization: Nearby Agent always encounters novel transition first.

Novel State Observation Activity: Active Novelty only occurs if the agent actually shoots an enemy.

Novelty Frequency: Less Frequent than Observation The agent can only shoot one enemy at a time, so there should be no more than one respawn per round.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is an event novelty, as the changes are situation-based, with clear entity-based preconditions. The respawn event cannot be environmental, because it relies on the relation enemy-dead, which is a result of player actions, and not an environmental relation.
- ✓ *Higher level novelties not present:* No higher level is defined.
- ✗ *Simpler novelty is possible:* It is possible to remove transformations from the sequence.
✗ **Possible remedy:** In a simpler event, the enemy might respawn in its last occupied position. The delay preconditions are required to differentiate this from an action novelty. However, we expect that moving the enemy back to its spawn point is less relevant to agent goals, and makes the novelty harder for an agent to learn.

SmartEnv

Target Level: 6 - Environment

Novelty Description: Temperature decrease causes motion sensors to trigger more often.
E/M/H based on amount of decrease.

Assumptions:

- Additional triggers are true positives (caused by actual motion events)
- All temperature changes occur prior to measurement / sensor triggers.

Discussion:

We model the effect of temperature decrease as an increased probability of detection for motions that would exist anyway.

Estimated Original T-SAL events

```
(:action shift-position
:parameters (?msize - real)
:precondition (and (eq (self) ?p)
                  (eq (person-position) ?pos)
                  (eq (current-time) ?t)
                )
:effect (and (motion-occurs ?pos ?msize ?t))
)
(:event sensor-triggers
:parameters (?s - sensor)
:precondition (and (motion-occurs ?mpos ?msize ?t)
                  (< (distance ?mpos (sensor-position ?s)) (sensor-radius ?s))
                  (< (sensor-threshold ?s) ?msize)
                  (< (last-trigger-time) ?t)
                  (eq (alert-probability) ?prob-alert)
                )
:effect (and (observe-trigger ?s) [?prob-alert]
            (set (last-trigger-time) ?t)
          )
)
```

Estimated T-SAL Transformation sequence:

```
ADDVALUEGENERATOR(AlertProb, UNIFORMDISTRIBUTION(EasyMin/EasyMax/MediumMax,
EasyMax/MediumMax/HardMax)
ADDERELATIONGENERATOR("alert-probability", NRELATIONDRAWS([], AlertProb, 1))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Not Present: No change in agent activity described.

Transition Novelty Type: None Only observations change.

Novelty Entry Determinism: None Assuming all temperature changes occur outside the time of measurement.

Novelty Obviousness: Statistical Novelty Only Unless the minimum time between events changes substantially, we can't consider observations to be new, just that their interarrival times are drawn from a new distribution.

Novelty Observation Determinism: Nondeterministic This depends on how much the agents move around, according to this model.

Novelty Observation Localization: Remote No novel transitions occur.

Novel State Observation Activity: Passive Can't take any actions.

Novelty Frequency: More Frequent than Observation Sensor triggers happen in continuous time, so this seems most apt.

Hierarchy Level Analysis

✓ *Analyzed level matches intended level:* This is environment novelty. The provided transformation sequence modifies only the distribution of the "alert-probability" relation, which is environmental as it considers no entities and is not affected by any agent action.

✓ *Higher level novelties not present:* Goals are clearly unaffected. Event novelty would require the creation of new sporadic events, which are presumably not intended here.

✓ *Simpler novelty not possible:* The above transformation sequence is minimal and cannot be reduced.

Target Level: 7 - Goals

Novelty Description: Inhabitant attempts to fool caregiver into thinking they went outside, but in fact they did not. This is based on fake leave_home/enter_home events inserted in the sensor stream. E/M/H based on number of fake events.

Assumptions:

- Fake leave_home and enter_home events are added such that real activities occur in between that the algorithm could detect normally.
- Event insertions are distributed randomly without respect to other user behaviors that could be used to predict them.

Discussion:

This is an ingenious way of modifying this domain to have goal novelties in it. We model the change to the event sequence as the result of policy only, not environment or task changes, with the exception of the goal itself.

Estimated T-SAL description of novel process:

(:metric LAZY-BUT-TRICKY-INHABITANT (- (time-spent-at-home) (apparent-time-spent-at-home))

Estimated T-SAL Transformation Sequence:

REPLACEPERFORMANCEEQUATION(LAZY-BUT-TRICKY-INHABITANT, '(- (time-spent-at-home) (apparent-time-spent-at-home)))

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Agent behavior changes to add trick events.

Transition Novelty Type: Modified Change Rate Leave_home and enter_home happen with a greater frequency than in original environment.

Novelty Entry Determinism: Infrequent nondeterministic These are nondeterministic because they occur according to an agent's simulated, and random "policy".

Novelty Obviousness: Novel Transition Discernible Assuming sensors don't have false positives when no one is at home, sensors should be triggered between "leave" and "enter" in the modified environment, but never the original. If such false positives happen at random times, this would be **statistical novelty only**.

Novelty Observation Determinism: None The observation function does not change.

Novelty Observation Localization: Nearby No novel observations.

Novel State Observation Activity: Passive Can't take any actions.

Novelty Frequency: Less Frequent than Observation Presumably, leave home and enter home events are much less frequent than sensor triggers.

Hierarchy Level Analysis

- ✓ *Analyzed level matches intended level:* This is clearly goal novelty.
- ✓ *Higher level novelties not present:* No higher levels apply, as no new event models are needed.
- ✓ *Simpler novelty not possible:* There is no simpler version of this transformation sequence that is also a novelty.

Target Level: 8 - Events

Novelty Description: Multiple days worth of data combined into one (fast forward). E/M/H based on number of days combined.

Assumptions:

- None known

Estimated Original T-SAL process

```
(:process time-marches-on  
:parameters ()  
:precondition ()  
:effect (increase (current-time) #t)  
)
```

Estimated Modified T-SAL process

```
(:process time-speeds-up  
:parameters ()  
:precondition ()  
:effect (increase (current-time) (* #t (- (combined-days) 1)))  
)
```

Estimated T-SAL Transformation Sequence:

```
ADDPROCESS(time-speeds-up)  
ADDPROCESSCHANGE(time-marches-on, '(increase (current-time) (* #t (- (combined-days) 1)))
```

Novelty Dimensions:

Novel External Agents: Novel External Agents Present: Agents will act much faster in the new version than previously, accomplishing activities in a fraction of the time.

Transition Novelty Type: Modified Change Rate All the same things happen faster.

Novelty Entry Determinism: None There's no observations before the fast forward.

Novelty Obviousness: Novel Transition Inferrable Assuming that certain activities, such as moving between rooms, have well-established minimum times, those activities occurring within a fraction of the time will cause observation sequences outside of the space of original environment observation sequences.

Novelty Observation Determinism: None The observation function does not change.

Novelty Observation Localization: Nearby No novel observations.

Novel State Observation Activity: Passive Can't take any actions.

Novelty Frequency: More Frequent than Observation Sensor triggers will occur more frequently, possibly more frequently than the observations.

Hierarchy Level Analysis

 *Analyzed level does not match intended level:* This appears to be an environment novelty, since it impacts everything the same way. In general, time-manipulation novelties seem to match the description of environmental novelty well. In our theory, this is a modification to a process with no preconditions, which counts as level 6. **Possible remedy:** A different event novelty might simulate sensor failures (e.g., by replacing truth results with random triggers or no triggers?) in certain entity-based situations (presence of multiple people? Too many or too few triggers over time?)

 *Higher level novelties not present:* No higher novelty levels are defined.

 *Simpler novelty not possible:* There is no simpler version of this transformation sequence that is also a legal novelty.