



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

**PREDICTING OPPONENT POSITION AND MODELING
UNCERTAINTY**

by

Kenneth J. Maroon

September 2020

Dissertation Supervisor:

Christian J. Darken

Research for this dissertation was performed at the MOVES Institute.

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the 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 Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2020		3. REPORT TYPE AND DATES COVERED Dissertation
4. TITLE AND SUBTITLE PREDICTING OPPONENT POSITION AND MODELING UNCERTAINTY				5. FUNDING NUMBERS
6. AUTHOR(S) Kenneth J. Maroon				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A				10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.				12b. DISTRIBUTION CODE A
13. ABSTRACT (maximum 200 words) Current combat simulation software developments for automated planning do not account for fog-of-war in their methods. This makes their outputs less realistic, as it is not reasonable to have the exact enemy positions in real-world planning. An artificial intelligence-controlled force should be able to operate without information that is not available to a human in the same situation. This dissertation presents a method for AI agents to predict and assess possible opposing force positions given typical intelligence products. We also present a method to aggregate the risk implications of these positions. We demonstrate the techniques in a combat simulation environment and evaluate their performance in multiple battle scenarios. The results show the importance of uncertainty in combat simulations and illustrate that our method of risk aggregation can be effective.				
14. SUBJECT TERMS modeling and simulation, automated planning, combat modeling, prediction modeling, risk assessment, artificial intelligence, AI, formation evaluation				15. NUMBER OF PAGES 171
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified		18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified		19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified
				20. LIMITATION OF ABSTRACT UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

PREDICTING OPPONENT POSITION AND MODELING UNCERTAINTY

Kenneth J. Maroon
Commander, United States Navy
BS, U.S. Naval Academy, 2001
MS, Modeling, Virtual Environments, and Simulation, Naval Postgraduate School, 2009

Submitted in partial fulfillment of the
requirements for the degree of

**DOCTOR OF PHILOSOPHY IN MODELING, VIRTUAL ENVIRONMENTS,
AND SIMULATION**

from the

**NAVAL POSTGRADUATE SCHOOL
September 2020**

Approved by: Christian J. Darken
Department of
Computer Science
Dissertation Supervisor

Jonathan K. Alt
Department of
Operations Research

Jeffrey A. Appleget
Department of
Operations Research

Imre L. Balogh
Department of
MOVES Institute

Arnold H. Buss
Department of
MOVES Institute

Christian J. Darken
Department of
Computer Science
Dissertation Chair

Approved by: Gurminder Singh
Chair, Department of Computer Science

Orrin D. Moses
Vice Provost of Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current combat simulation software developments for automated planning do not account for fog-of-war in their methods. This makes their outputs less realistic, as it is not reasonable to have the exact enemy positions in real-world planning. An artificial intelligence-controlled force should be able to operate without information that is not available to a human in the same situation. This dissertation presents a method for AI agents to predict and assess possible opposing force positions given typical intelligence products. We also present a method to aggregate the risk implications of these positions. We demonstrate the techniques in a combat simulation environment and evaluate their performance in multiple battle scenarios. The results show the importance of uncertainty in combat simulations and illustrate that our method of risk aggregation can be effective.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
1.	AI Uses in Combat Simulation.....	2
2.	“Cheating” Distortions	3
B.	PROBLEM STATEMENT	7
C.	OBJECTIVES	7
D.	DOCUMENT OUTLINE	8
II.	BACKGROUND	9
A.	APPLICABLE AI TECHNIQUES AND EXAMPLES.....	9
1.	Space-based Methods for Opponent Tracking.....	9
2.	Opponent Tracking with Particle Filters.....	11
3.	Tactical Annotation and Formations	14
4.	Prediction in Video Games.....	19
5.	Prediction in Military Applications.....	22
6.	Summary.....	22
B.	THE MILITARY DECISION-MAKING PROCESS	23
C.	WOMBAT XXI.....	26
1.	A Simpler Environment for Faster Prototyping.....	27
2.	Entity Representation	28
3.	Terrain and Risk Representation	29
4.	Combat Simulation	30
5.	Data Collection	31
III.	METHOD	33
A.	ASSUMPTIONS.....	33
B.	FORMATION SCORING FUNCTION	34
1.	Preferred Targets and Field of Fire	35
2.	Distance to Unit Leader.....	35
3.	Distance to Other Entities	35
4.	Lines of Fire.....	36
5.	Cover	36
6.	Weighting.....	36
7.	Scoring Function Algorithm	37
8.	Building the Scoring Function	38
C.	PREDICTION DEFINITION.....	38
D.	RANDOM PLACEMENT/HILL CLIMB.....	38

1.	Randomization	39
2.	Hill Climbing	40
3.	Hill Climbing Algorithm	40
E.	HANDLING LOCAL MAXIMA.....	41
F.	CANONICAL ENSEMBLE.....	42
1.	State Probability.....	43
2.	Partition Function	43
3.	Energy	43
4.	Entropy	43
G.	GIBBS DISTRIBUTION.....	44
1.	Formation Probability	44
2.	Distribution Entropy	44
3.	Aggregated Risk Mapping	45
H.	PLAN RECOGNITION	46
I.	SUMMARY	47
IV.	IMPLEMENTATION	49
A.	CONFIGURING AND MODIFYING WXXI	49
1.	Terrain	49
2.	Units and Entities.....	50
3.	VisibilityGraph and VisibilityGraphPre	53
4.	Predictor Class	55
B.	FORMATION SCORING FUNCTION	57
1.	Preferred Target and FOF	57
2.	Distance to Unit Leader.....	58
3.	Inter-Entity Spacing	58
4.	Lines of Fire.....	59
5.	Cover	60
C.	HILL CLIMBING	61
D.	GIBBS DISTRIBUTION.....	66
1.	Building the Candidate Formation Set Distribution	66
2.	The Uncertainty Threat Graph	70
3.	Entropy Effects.....	73
E.	PLAN RECOGNITION	76
F.	INTEGRATION WITH OTHER MODULES.....	80
G.	SUMMARY	81
V.	EVALUATION	83
A.	COMPONENT TESTING	83
B.	BATTLE SCENARIO	84

1.	Scenario Modifications	85
2.	Scenario Force Structure and Armament	86
3.	Simulator Output and Data Collection	86
C.	FORMATION SCORING EVALUATION	89
1.	Scenario and Formation Score Specifics.....	89
2.	Results/Analysis.....	92
3.	Summary.....	98
D.	UNCERTAINTY THREAT GRAPH EVALUATION	98
1.	Scenario Specifics.....	99
2.	Blue Planning for Only Visible Enemies.....	100
3.	Blue Planning with Estimated Risk.....	103
4.	Blue Planning with Perfect Information.....	106
5.	Battle Results and Analysis.....	107
6.	Summary.....	110
E.	BLUE PLANNING WITH FOG-OF-WAR	110
1.	Deliberate Attack Verse Movement to Contact	110
2.	Scenario Specifics.....	111
3.	The Fog-of-War Engagement	113
4.	Updating with Estimated Risk.....	118
5.	Fog-of-War Conclusions.....	119
F.	SUMMARY	120
VI.	CONCLUSIONS	121
A.	RESEARCH CONCLUSIONS AND CONTRIBUTIONS	122
1.	Enhancing Combat Simulation by Adding Fog-of-War.....	122
2.	The Gibb Distribution and Aggregated Risk Estimate	123
3.	Encoding the IPB as Formation Score	123
B.	FUTURE WORK.....	124
APPENDIX A. THE PREDICTOR EDITOR.....		127
APPENDIX B. THE PLAN PREDICTOR EDITOR		137
LIST OF REFERENCES.....		141
INITIAL DISTRIBUTION LIST		145

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	2012 League of Legends Stage Set-Up. Adapted from Drain (2012).....	4
Figure 2.	The Xenomorph. Source Thompson (2017).	6
Figure 3.	Occupancy Map Example. Source: Isla and Blumberg (2002).....	10
Figure 4.	Occupancy Map and Particle Filter Example. Source Darken and Anderegg (2008).	12
Figure 5.	Particles Moving According to Simulacra and Constrained to a Navigation Graph. Source Darken and Anderegg (2008).....	13
Figure 6.	Tactical Visibility Example. Source: Linden (2002).	14
Figure 7.	Identifying Possible Ambush Locations (Grey Nodes). Source: Linden (2002).....	16
Figure 8.	Tactical Waypoint Scoring. Adapted from Straatman et al. (2005).	18
Figure 9.	Translating Real Fireteam Movements Into a Simulation. Source Darken, McCue, and Guerrero (2010)	19
Figure 10.	The EISBot Playing Starcraft. Source Weber, Mateas, and Jhala (2011).....	21
Figure 11.	The MDMP. Source: U.S. Army (2015).....	24
Figure 12.	Adversary Template. Source: U.S. Marine (2014).	25
Figure 13.	Example of a Situation Template. Source U.S. Marine (2014, 4–9, Figure 4–6).....	26
Figure 14.	WXXI Implementation in Unity3D.	28
Figure 15.	WXXI Entity and Hierarchy	29
Figure 16.	WXXI Example Showing the Terrain-based Triangle-mesh Navigation Graph. Source Harder (2017).	30
Figure 17.	Area of Operation Example.	39
Figure 18.	The Cayucos Creek Terrain. Source Harder (2017).	50
Figure 19.	FOF Designation.	52

Figure 20.	Line of Fire Designation.	53
Figure 21.	The Predictor GameObject (Blue Box) and Editor.	56
Figure 22.	Formation Scoring Function Components.	60
Figure 23.	Blue VisibilityGraph For Cover Calculations.....	61
Figure 24.	Simple Hill Climbing Example Starting Positions.	62
Figure 25.	Simple Hill Climbing Example Ending Positions.	63
Figure 26.	Complex Hill Climbing Example Starting Positions.	64
Figure 27.	Complex Hill Climbing Example Ending Positions.	65
Figure 28.	Results From Building the Distribution.	69
Figure 29.	VisibilityGraphs for Formations in the Distribution.....	71
Figure 30.	Uncertainty Threat Graph for Entropy = 0.9999.	73
Figure 31.	Distribution Results for Entropy = 0.5.....	74
Figure 32.	Uncertainty Threat Graph for Entropy = 0.5.	75
Figure 33.	The Plan Predictor Editor.....	77
Figure 34.	Box Formation Scoring Function Assignments.	78
Figure 35.	Ambush Formation Scoring Function Assignments.	79
Figure 36.	Ridge Defense Formation Scoring Function Assignments.	80
Figure 37.	The Terrain Reasoning Module.	81
Figure 38.	WXXI Combat Example.	87
Figure 39.	WXXI Combat Example With VisibilityGraph.....	88
Figure 40.	Formation Scoring Evaluation Scenario.	90
Figure 41.	Formation Score Penalty Weights	91
Figure 42.	Blue VisibilityGraph for Cover Calculations.	92
Figure 43.	Scatterplot of Blue End Strength versus. Formation Scores for All Formations.	93

Figure 44.	Cohesion Penalty Example.	94
Figure 45.	Blue End Strength versus Formation Scores (Outliers Removed).....	95
Figure 46.	Average Red Survival Rate versus Formation Scores (Outliers Removed).....	95
Figure 47.	Average Mission Accomplishment Score versus Formation Scores (Outliers Removed).....	96
Figure 48.	Average FER versus Formation Scores (Outliers Removed).	96
Figure 49.	Uncertainty Threat Evaluation Scenario	99
Figure 50.	VisibilityGraph For Detected Enemies.	101
Figure 51.	Attack Plan for Observed Entities.....	102
Figure 52.	FOF Assignments and Cover Calculation VisibilityGraph.	103
Figure 53.	Uncertainty Threat Graph form 50 Formations with Entropy = 0.70.	104
Figure 54.	Plan of Attack created with Uncertainty Threat Graph.	105
Figure 55.	Perfect Information VisibilityGraph with Blue Pathfinding.....	106
Figure 56.	Box Plots of Average Blue End Strength.	108
Figure 57.	Box Plots of Average Red Survival Rate.....	108
Figure 58.	Box Plots of Average FER.....	109
Figure 59.	Fog-of-War Evaluation Starting Conditions With Zeroed VisibilityGraph.	112
Figure 60.	Initial Blue Movement Plan in Fog-of-War Scenario.....	114
Figure 61.	Updated Detected Enemies VisibilityGraph and Pathfinding.	115
Figure 62.	Blue Detects More of Red's Entities.	116
Figure 63.	Second Update to Detected Enemies VisibilityGraph.	117
Figure 64.	Updated Pathfinding.	117
Figure 65.	Fog-of-War Uncertainty Threat Graph.	118
Figure 66.	Updated Pathfinding with Uncertainty Threat Graph.	119

Figure 67.	Predictor Editor Top.	127
Figure 68.	Predictor Editor Bottom.....	133
Figure 69.	Plan Predictor Editor.....	137

LIST OF TABLES

Table 1.	Random Placement and Hill Climbing Test Results. Source: Maroon and Darken (2019).	84
Table 2.	Battle Indicator Results for Uncertainty Threat Graph Evaluation.	107

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

2D	two-dimensional
AI	Artificial Intelligence
COA	Course of Action
COMBATXXI	Combined Arms Analysis Tool for the 21 st Century
CSE	Combat Simulation Environment
CSV	comma-separated-value
DES	discrete event system
ECOA	Enemy Course of Action
FER	fractional exchange ratio
FOF	field of fire
HTN	Hierarchical Task Network
IPB	Intelligence Preparation of the Battlespace
MDMP	Military Decision Making Process
OP	observation post
OTB	OneSAF-Test Bed
PUBG	PlayerUnknown's Battlegrounds
RAID	Real-time Adversarial Intelligence and Decision Making program
RTS	real time strategy game
SOP	standard operating procedures
TSM	Team Solomid
WXXI	WOMBAT XXI

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

In integrating artificial intelligence (AI) techniques with combat simulations, it is important that fog-of-war considerations are included and accounted for. Current approaches for automated planning depend on perfect knowledge of opposing force positions, making the actions their AI-controlled agents less realistic. In order to make more natural agent behavior, the AI force should receive and act on the same type and amount of information available to human commanders in a similar situation. This will require the AI to depend on an understanding of its opposition and environmental conditions, just as a human would.

This dissertation focuses on two aspects of AI's ability to handle fog-of-war in a simulation environment. The first aspect is the human representation of knowledge about the enemy and environment presented to the AI. This includes the enemy priorities and procedures, as well as environmental considerations such as visibility and sight lines. The second aspect is when given this knowledge, the AI can predict likely enemy positions in the fog-of-war and assess the risk implications for probable enemy formations.

In regard to the first aspect, we reviewed AI methods for opponent tracking and tactical annotation within a simulation environment. We identified the need to discretize the combat terrain into a navigation graph. This navigation graph would have edges to support entity movements and nodes that could be annotated, providing strategic information to the AI agents. We also identified techniques for the tactical evaluation of unit geometry in a combat simulation environment.

We then found it applicable to examine the analysis conducted during the Military Decision Making Process (MDMP), and specifically, the products produced by the intelligence preparation of the battlespace (IPB). The output of the IPB includes a layering of known adversary tactics onto the current operational environment to produce an expected Enemy Course of Action (ECOA), as seen in the situation template depicted Figure 1.

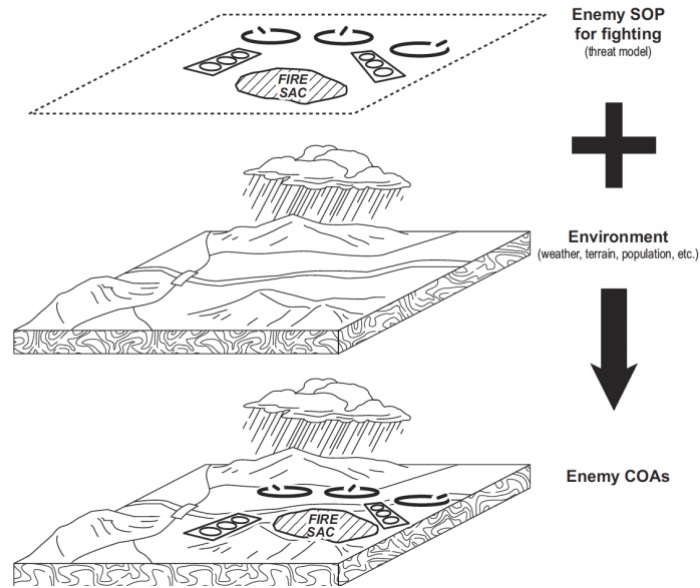


Figure 1. A Situation Template Example. Source U.S. Marine Corps (2014).

The tactical annotation of nodes can be seen as the top layer enemy threat model shown in Figure 1. When these annotations are applied to the navigation graph of the terrain (the environment layer), the result is an encoding of the ECOA into the simulation. We can then evaluate how well potential enemy positions align with the encoded ECOA.

We called this evaluation the formation scoring function. This scoring function is the starting point for allowing the AI to predict and assess risk in the fog-of-war. The function produces a numerical score that depends on a given enemy formation's ability to target objective priorities while still maintaining a desired unit geometry and cover from fire. Higher scores indicate more conformity with the encoded ECOA. We called a formation that maximized a given scoring function a prediction.

Unfortunately, identifying a maximizing formation is not a trivial task, since it involves multiple entities balancing several competing factors to produce a score. We used the approach of randomly placing undetected entities in likely locations on the terrain and letting them hill climb to positions that maximized their contribution to the formation score. With several rounds of random placement and hill climbing, candidate set of formations is produced, with each formation being a local maximum of the scoring function. Each

formation consists of a different set of enemy entity positions and associated threat values for the nodes in the navigation graph.

We then use Gibb's canonical ensemble equations, treating each formation in the candidate set as a possible "state" that the actual enemy formation could be in. This allows us to calculate a probability value for every formation in the candidate set. We can then aggregate the node threat values from each formation according to its calculated probability. This enables the representation of the estimated risk for the probable enemy positions as a separate graph we call the uncertainty threat graph. Since the graph is produced only with available intelligence products and observations, it is not reliant on extraneous information that human commanders would not have access to.

This dissertation presents our implementation and evaluation of these methods in WOMBAT XXI (WXXI), a lightweight combat simulation environment constructed in the Unity 3D game engine (Figure 2).

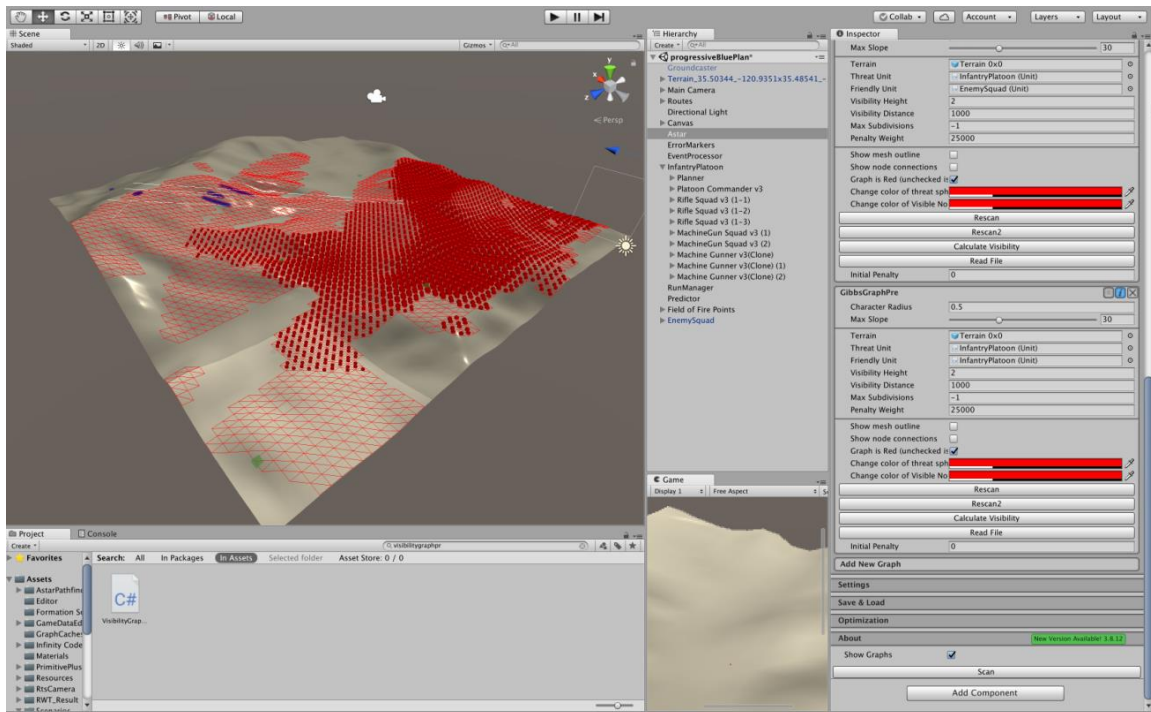


Figure 2. A WXXI Scene in Unity3D.

Using WXXI's unit templates and real-world terrain representation, we conducted 60 simulated battles across three different levels of information about enemy force positions. The battles simulated attacking a blue platoon against a static red squad. In the first scenario, we limited blue's knowledge to a set of detected red defenders. For the second scenario, we used the detected red defenders and a formation scoring function to produce an uncertainty threat graph to estimate the actual red formation. For the final scenario, blue was given full information about the red positions. In each battle, we collected data on blue and red casualties, fractional exchange ratios (FER), and mission accomplishment. We found that when blue was provided with our estimate of red's formation, they performed comparably to the when they were given the actual positions. Table 1 provides the quantitative data collected on the three battle scenarios.

Table 1. Battle Testing Results by Scenario.

Scenario	Attacker (Blue) End Strength		Defender (Red) Survival Rate		Defender/Attacker FER		Mission Accomplishment Score	
	μ	Std. Err	μ	Std. Err	μ	Std. Err	μ	Std. Err
Detected Enemies Graph	32.50	1.00	0.79	0.02	0.56	0.06	0.70	0.11
Uncertainty Risk Graph	47.50	0.63	0.78	0.01	2.28	0.32	1.00	0.00
Perfect Information Graph	51.35	0.40	0.93	0.00	1.92	0.28	1.00	0.00

The dissertation's novel approach to prediction and uncertainty modeling has great potential for applications in automated planning and for improving AI-controlled forces for wargaming. The dissertation also provides further testing examples for movement to contact operations and provides a method for plan recognition using the formation scoring function. Finally, we contribute added functionality to the WXXI combat simulation environment.

Reference

U.S. Marine Corps. 2018. *Infantry Company Operations*. MCRP 3-10A.2. Washington, DC: Headquarters United States Marine Corps.
<https://www.marines.mil/News/Publications/MCPPEL/Electronic-Library-Display/Article/900521/mcrp-3-10a2-formerly-mcrp-3-10a1/>

ACKNOWLEDGMENTS

I am happy to dedicate this dissertation to my wife, Aisha, and children, Sophia and Cole. Aisha's hard work in extremely difficult times during this process (and through the entirety of me knowing her) enabled me to focus on my dissertation and accomplish my goals. She made numerous sacrifices to ensure our family was taken care of through quarantining, distance learning, and moving in the middle of all of it. My kids were a great source of motivation (and sometimes distraction). I am grateful that they let me work when I needed to but also that we got to enjoy the beauty of Monterey together.

I would especially like to thank my committee chair and advisor, Chris Darken, for keeping me on task through every weekly meeting and Skype call. I deeply value your mentorship and the knowledge about AI you passed to me.

Many thanks to my committee members for their time, support and insight. Imre Balogh, Arnie Buss, Jeff Appleget, and Jon Alt made the finest committee any student could ask for, and I appreciate their guidance through the testing, writing, and defense process.

I would like to thank the faculty and students in the MOVES Institute. I am always amazed by the hard work and ingenuity that comes from all of the incredible people in Watkins Hall (and across the whole campus). I have been truly blessed to be able be a part of MOVES at two different points in my life and look forward to future collaborations.

Finally, thanks to Jane and Bill Maroon for always pushing me to do better and reach higher. I would not be here today without your love and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Artificial intelligence (AI) competitors have had some success in video game “battlefields,” and it seems reasonable that techniques used in AI development can be useful in military simulations and planning automation. In December 2018, AlphaStar, an AI designed to play the RTS *StarCraft 2* was able to beat a top-level human opponent under professional match conditions (AlphaStar 2019). *StarCraft 2* is a real-time strategy (RTS) game that can have a large number of entities in the simulated battlefield at any time. This victory was significant because RTS games present a hard problem for AI developers, and *StarCraft 2* is one of the most popular games in the genre. Unlike chess or Go, RTS features a wider variety of units, technology research trees, economic management, and imperfect knowledge of the opposition. AlphaStar’s success is impressive in that it is accomplished without giving the AI any information that a human player would not receive. The fact that an AI could succeed in a conceptualized battlespace demonstrates how it would be beneficial to apply AI techniques to combat simulation.

For human commanders, the knowledge of the enemy positions is aided by planning staffs conducting a proper Intelligence Preparation of the Battlespace (IPB). The IPB process entails analysis of a multitude of products relating to weather, terrain, enemy capabilities, and possible enemy locations. The importance of the IPB to a commander’s ability to understand the enemy is highlighted by James et al:

Behavior of a uniformed enemy is somewhat predictable because it is based on doctrine, movement rates, and the limitations of weapon systems and on the interaction of system components that are not individually purposeful. Observation of a single system component can lead to reasonable conclusions about the behavior of the larger system. This knowledge state is amenable to the present-day IPB. (2017, 2)

While the amount of intelligence and data available to command staff has grown considerably over the years, the time to analyze it and generate feasible Courses of Action (COAs) has not increased. Since the IPB is a thorough examination of the battlespace, it can become very manpower-intensive and time-consuming. Edward Ballanco of the Army

War College notes that “staffs today have more intelligence products . . . but no enemy analysis tool [that] analyzes terrain, weather, and enemy weapons and creates an optimal enemy course of action” (2019). He also notes that current AI advancements, particularly in real-time strategy (RTS) games, make the development of this type of tool more viable.

1. AI Uses in Combat Simulation

Current combat simulation software developments for automated planning do not account for the fog-of-war in their methods. Recent work by Harder (2017) and Severson (2019) have used AI methods to automate fire support and maneuver planning. However, both of their approaches depend on perfect knowledge of the enemy force locations. This makes the outputs of their respective methods less realistic. It is not reasonable to expect to have the exact enemy positions in real-world planning. An AI-controlled force should be able to operate without information that is not available to a human in the same situation. Being able to operate in some level of fog-of-war induced uncertainty will give AI-controlled entities behavior that is both more dynamic and realistic. Rather than moving to engage an enemy at an exact, perfectly known location, the AI forces will have to rely on their understanding of enemy tactics and terrain, just as a human commander would.

Humans use their experience, understanding of the local terrain, environmental conditions, and enemy intelligence estimates to make informed decisions about where they believe their opponents are. Simulation designers attempting to have bots make similar predictions will often use scripted behaviors or give the bots additional data not available to human participants (Straatman, van der Sterren, and Beij 2005). (The language in the previous two sentences, as well as other contextual materials in this dissertation, was repurposed from Maroon and Darken 2019). However, giving the bots extra knowledge can be detrimental to the overall simulation. The bots can target units or respond to events that should not be seen in the game’s fog-of-war. This creates behaviors that are not natural or feel like “cheating.” Battles are rarely ever fought perfectly, and we would expect bots to make mistakes in line with a human in a similar situation with associated effects of the fog-of-war. An intelligence-based prediction model will make the AI decisions more consistent with the choices a typical commander would make.

2. “Cheating” Distortions

When one side in a battle knows opposing positions that they should not have, the change in their tactics and behavior can be significant. Examples of how this distortion affects the user experience can be seen in cheating incidents that have occurred several video game battles. It is important to note that all of these cheats were used to remove the fog-of-war piece of their respective games. In removing this element, the behavior of the cheating player, or AI, was altered in an unrealistic way.

The first example shows how a professional RTS team was able to avoid an ambush due to their unfair knowledge of opponent positions in a championship tournament. *League of Legends* is a multiple player battle arena game that is often played in front of huge crowds. The 2017 World Finals filled the 40,000-seat Beijing Olympic Stadium (Pei 2019). In these events, an enlarged playing field is projected on to screens for the viewing spectators. These projections often have information not made available to the competitors in order to allow spectators to better follow the action. At the 2012 World Finals held at Los Angeles, CA, the competitors were seated in front of the spectator display, shown in Figure 1.



Team Solomid's mini-map is circled in red. Azubu Frost was able to use this to gain position information about their opponent.

Figure 1. 2012 League of Legends Stage Set-Up. Adapted from Drain (2012).

This seating arrangement allowed for some players to see a mini map of the battlespace that showed all player positions. This is exactly what happened in the semi-final match between teams Azubu Frost and Team Solomid (TSM).

In the first game against Azubu Frost, TSM went for an early top-lane kill, sending the full team to top at the very start. In a fairly damning photo . . . you can see one of Frost's players with his head turned to look across and upward across the stage, where you could see 5 TSM icons moving north along the top lane. A moment later, Azubu Frost put a warning ping on the top lane. (Zacny 2014)

In addition to the video evidence of players looking back, Team Solomid was attempting a non-standard opening which should have caught Azubu Frost out of position (Zacny 2014).

The ability to unrealistically react to an unexpected change in playstyle by TSM lead Azubu Frost to a victory, but their tactics were later deemed cheating by tournament officials (Drain 2012).

In a separate event, multiple professional competitors in *PlayerUnknown's Battlegrounds* (PUBG) were banned after it was revealed that they were using Radar Hacking to get player positions. PUBG is a battle royal style multiplayer shooter. Radar hacking involves taking server information from the game and sending it to a separate device (Groux 2018). The cheating player can then use the device to see in-game positions of opponents. While Radar Hacks are usually caught by anti-cheat algorithms, players using them can often be identified by in-game play that is too perfect. In all, 30,000 player accounts were banned (Groux 2018).

The final “cheating” example comes not from esports but comparing the two AIs found in the game *Alien: Isolation*. The game is based on the Ridley Scott movie *Alien* and involves the player being chased by a single xenomorph (the alien monster seen in Figure 2) through a mostly abandoned space station. The xenomorph can never be killed, and players spend most of the game trying to hide from it.



Figure 2. The Xenomorph. Source Thompson (2017).

The behavior of the xenomorph is controlled by two distinct AI implementations (Thompson 2017). The first AI is the high-level director AI that has exact knowledge of the player position. The second lower-level AI is assigned directly to the alien and is not given the player location. It can search for the player using the xenomorph's sense of sight and hearing. In the game, the director AI controls the encounter rate of the xenomorph. It will guide the alien towards the player when the game wants to create tension and away from the player to allow time to accomplish in-game objectives safely. The xenomorph is never passed the exact player position and once guided to an area, must hunt the player using its own AI. When both AI are working in balance the game is terrifyingly fun. Watching the xenomorph's actions (from a safe hiding spot), it really feels like you are in a fair battle against an actual opponent. However, if the director AI provides too much guidance to the xenomorph, the alien seems to "cheat," moving right on top of a perfect hiding spot or patrolling the exact area a player is trying to move to. In these cases, the alien behavior appears broken and unnatural.

As we look to develop AI implementations for combat simulations, we need to ensure that the behaviors remain consistent and natural. To better accomplish this, we need

to ensure that our AI methods can be implemented in information-constrained environments.

B. PROBLEM STATEMENT

While applying AI techniques to combat simulation presents a number of issues to consider, this dissertation focuses on the problem of developing a method for an AI-controlled force to predict opponent positions based on environmental data and intelligence information available to humans. Additionally, human commanders often have multiple locations where they believe enemy forces may be. Since the AI cannot be certain of a single disposition of enemy forces, this dissertation will present a distribution that allows the AI to assess multiple possible predictions. By assigning probabilities to multiple predictions according to our distribution, it will be possible for the AI to calculate an expected threat value for any location in the simulated battlefield.

C. OBJECTIVES

The main objective of this dissertation is to develop the ability for AI agents in a Combat Simulation Environment (CSE) to predict and assess possible opposing force positions given information typically available to human commanders. Once agents have a method for determining probable enemy positions, they will need to be able to aggregate the risk implications of these positions. This will mirror a human estimating risk from multiple threat vectors. In doing so, we hope to make agent behavior more realistic within the CSE.

In order to accomplish this main objective, this dissertation outlines the development of the following:

- a method to apply enemy tactics, spacing, and objectives to a given terrain.
- a scoring function to evaluate an enemy formation given their applied tactics.

- a method to identify enemy formations that maximizes the scoring function.
- a method to assign a probability for each formation in a set of possible formations according to its formation score. This is the probability that the formation is the actual formation.
- a method to calculate the aggregated risk effects of all possible formations when given a set of formations with an associated probability.

D. DOCUMENT OUTLINE

The dissertation comprises the following six chapters and three appendices. Chapter II reviews background material relevant to AI methods for opponent tracking and tactical movement. It also provides information on the Military Decision Making Process (MDMP), with a particular emphasis on the Intelligence Preparation of the Battlespace (IPB). Finally, the chapter describes WOMBAT XXI (WXXI) a CSE that was used to develop and test the methods in the dissertation. Chapter III presents the actual methods and provides background information on the canonical ensemble and the Gibbs distribution, which are concepts that are directly applicable approach. Chapter IV details the implementation of the methods in WXXI. Chapter V describes evaluations of our implementations using the WXXI combat simulation. Chapter VI lays out the conclusions and contributions of the dissertation and provides recommendations for future research. Finally, we have included two appendices to highlight the new WXXI modules. These appendices are included to discuss specific additions made to WXXI during the development and testing of our prediction methods. Appendix A describes the Predictor Class Editor. Appendix B describes the Plan Predictor Editor.

II. BACKGROUND

In this chapter, we present foundational techniques and methods we will use in our approach. We review AI algorithms for opponent tracking, tactical annotation, and formation evaluation. Elements from the practices presented in this chapter will be employed by our methods in Chapter III. We also provide examples of opponent prediction in video games and military applications. In addition, the chapter reviews the MDMP and presents products from the IPB that will be used by our predictor. The chapter ends with a summary of the relevant features of WXXI, the CSE used extensively in our research.

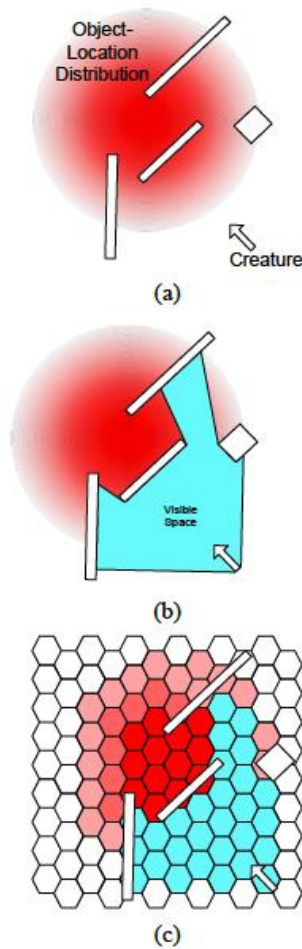
A. APPLICABLE AI TECHNIQUES AND EXAMPLES

There is a rich body of work regarding realistic opponent position tracking and tactical considerations from which a prediction model can be built. We start by looking at two distinct methods for following an entity as it moves out of visual range. Our predictor methods share many of the same requirements found in the opponent tracking algorithms. They both involve an understanding of the surrounding environment and a probability assignment for possible occupied locations.

1. Space-based Methods for Opponent Tracking

Tozour introduced space-based methods for opponent tracking using influence maps. Isla and Blumberg developed tracking methods employing occupancy maps. Each of these methods required that the search space be discretized into a directed graph. Each graph node represents a specific section of the search space and is connected only to nodes with which shares an edge. Tozour (2001) described influence mapping as the “AI agent’s knowledge about the world” (287). Influence maps can be used to store a wide array of game information about each node, including (but not limited too) visibility, available resources, terrain features, walkability, and present entities. Similar to influence mapping, Isla and Blumberg’s occupancy maps also have nodes but each is assigned a probability for being occupied by an opponent (Isla and Blumberg 2002). Nodes that are visible and empty are assigned a zero probability. The expected probability that an opponent is in a given node is the node's probability value divided by the sum of all of the graph’s nodes’

probability values (Isla and Blumberg 2002). Figure 3 illustrates an example of an occupancy map.



(a) Shows the Gaussian distribution of possible locations. (b) Updates the distribution based on visible areas. (c) Discretizes the space into nodes, with the darker red having a higher probability of occupancy (Isla and Blumberg 2002).

Figure 3. Occupancy Map Example. Source: Isla and Blumberg (2002).

Probability is diffused between nodes over time with each node passing some of its probability value to the nodes it is connected to. Diffusion of probability can be done using isotropic diffusion or based the on the opponent's last observed motion (Isla and Blumberg 2002). Node probability values are also updated as new nodes become visible through

culling. This means that if a node becomes visible and is observed to be empty, its probability value is reset to zero (Isla and Blumberg 2002).

There are some issues with occupancy maps. The precision of the opponent position prediction is dependent on the granularity of the directed graph. This is because the occupancy map only gives the probability that a node can be occupied. It does not say where in the node the opponent is. This can be a problem for graphs with less, larger sized nodes. While this can be solved with an increase in the number of nodes, it will also increase computation and memory costs (Darken and Anderegg 2008). Also, the diffusion of probability may not reflect the actual speed and motion that the opponent actually moves at (Darken and Anderegg 2008).

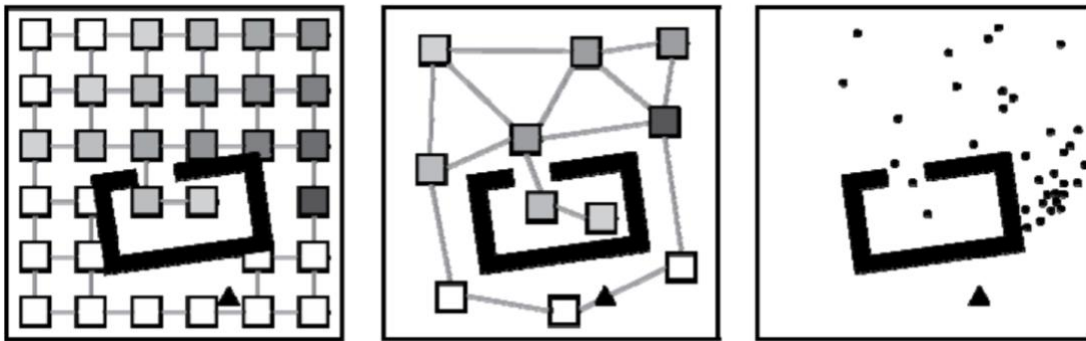
2. Opponent Tracking with Particle Filters

Alternatively, Bererton (2004) proposed the use of particle filters to reflect possible positions as a finite set of weighted samples that move randomly in the search space. Unlike the nodes of occupancy maps, particles represent a single position in the search space. Similar to the nodes, each particle is assigned a probability, or weight, that it is the opponent's actual position. The expected probability that a particle is the exact position of the opponent is the particle's probability weight, divided by the sum of the weights of all the particles (Bererton 2004).

Each particle is updated independently of the other particles in the system. Particles randomly select a direction and then try to move at the maximum velocity of the opponent (Darken and Anderegg 2008). Particle movement is subject to the search environment and is adjusted if there is an obstacle (such as a wall) that the opponent would not be able to move through. If the particle moves to an observable area it is removed from the filter (Darken and Anderegg 2008).

Particle filters can be more precise than an occupancy map, as they represent positions, not regions. Figure 4 provides examples of occupancy maps and a particle filter from Darken and Anderegg (2008). It shows how the precision of occupancy maps is dependent on the number of nodes. The left and center squares show occupancy maps at different granularity levels. The right square is an example of a particle filter. The lower

resolution occupancy map is unable to capture the probability that the opponent could be at the right edge of the box because there is no node in that area that is not visible to the tracking AI. Additionally, particles move at a speed that is more relatable to the opponent being tracked and not by diffusion rates as in occupancy maps. However, their movement is still dependent on a random walk and does not represent tactical movement (Darken and Anderegg 2008). Particle filter computation costs are related to the total number of particles in the filter and will rise as the number of particles increases but not as the size of the search space grows larger. There is also a computational cost for calculating the movement of the particles, particularly collision testing with objects in the search space.

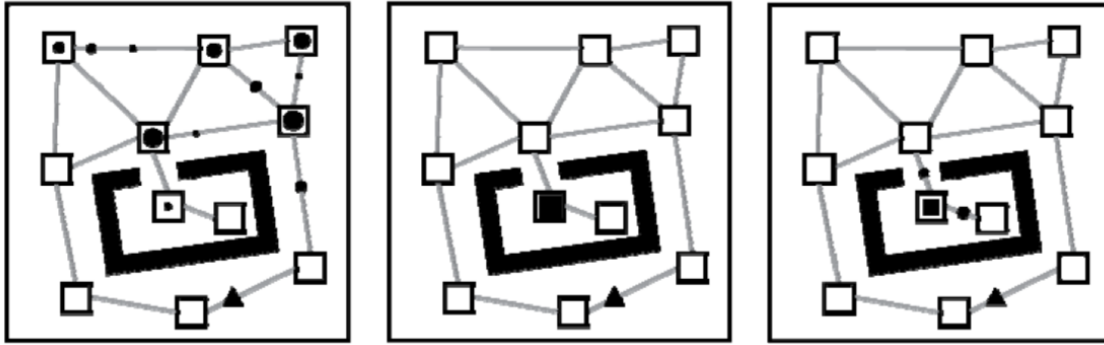


Note the differences between occupancy maps (left and middle) and a particle filter (right). In all examples, the agent, represented by the black triangle is tracking an opponent who disappeared around the right side of the box.

Figure 4. Occupancy Map and Particle Filter Example. Source Darken and Anderegg (2008).

Darken and Anderegg further refined particle filters by using defined motion models called “simulacra” for their particles, producing predictions that can reflect different behaviors expected in opponents (2008). This allows their predictions to account for tactical movement by the opponent. They also combined both approaches by having their particles move according to a set navigation graph in the search space. This reduces the computation costs for particle movement and allows them to store locations for path planning (2008). An example from their work is shown in Figure 5, which has three examples of particles moving on a navigation graph according to different simulacra

(2008). The first square shows particles moving in a random-walk. Note that while the navigation graph has the same nodes as the sparse graph (middle) in Figure 4, it can capture some probability that the opponent could be at the right edge of the box. This is because the particles can populate the edges that connect the nodes, not just the nodes themselves. The second square has particles moving to nodes that provide cover. The last square shows particles first moving to cover, then performing a random-walk. This example shows how the precision of the prediction is improved with the application of some tactical considerations



In all three examples, circle particles are moving according to a random walk. Square particles move to nodes that provide cover. Particle size indicates weight (Darken and Anderegg 2008).

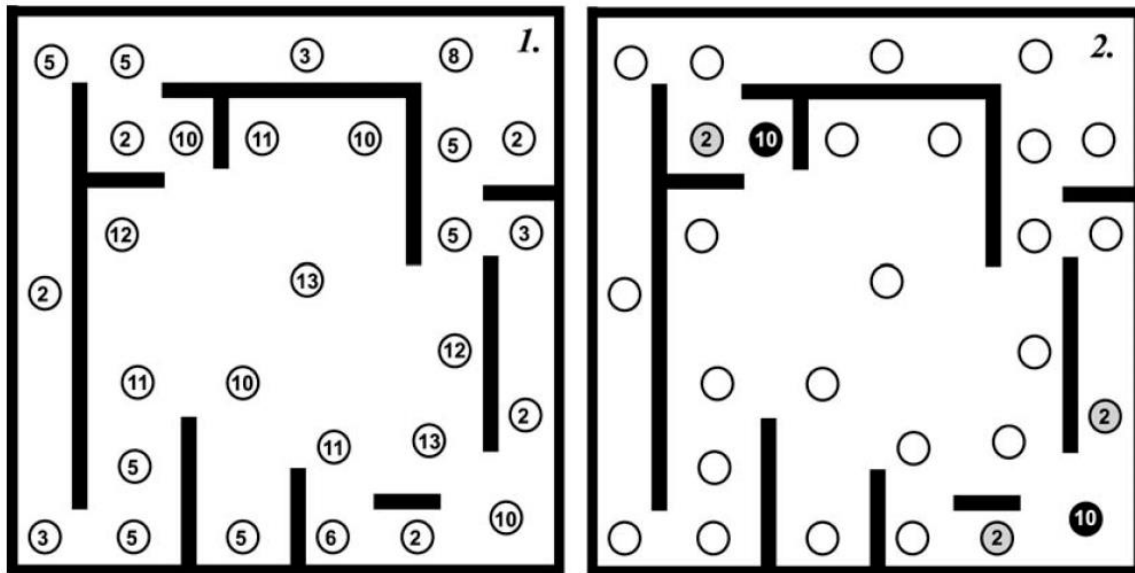
Figure 5. Particles Moving According to Simulacra and Constrained to a Navigation Graph. Source Darken and Anderegg (2008).

Similar to both of the opponent tracking techniques presented in the previous sections, we recognize the need for a defined discretization of the environment in our prediction methods. The node structure of a navigation graph balances the need for fidelity with computational costs. The previous techniques relied on diffusion or particle movement to assign probabilities of opponent locations. Our methods determine probable opponent positions based on an evaluation of the annotation of nodes in the navigation graph and tactical entity spacing.

3. Tactical Annotation and Formations

In order to improve our positional evaluations, we looked at AI techniques for encoding tactical behaviors. It seems reasonable that if we want our methods to make a tactical decision, we should have an understanding of how tactics are represented in a simulated environment.

Liden (2002) developed methods to identify strategic points of interest in a simulation by analyzing the navigation graphs available to the AI. He advocated for precalculating visibility between graph nodes to support this analysis. Once the visibility of each node is known, cover and attack positions can be established. Nodes that are highly visible present good attack opportunities but poor cover, and nodes with low visibility allow for safety but limit attack options (Liden 2002). Figure 6 shows a further step of this analysis by identifying nodes that have high visibility and neighbor nodes with significantly lower visibility. These are good locations to fire from and then retreat to cover (Liden 2002).



Node numbers represent visibility, or how many other nodes are seen (or can see) the node. The square on the right identifies good attack squares as nodes that have high visibility (for attack) next to low visibility nodes (for cover).

Figure 6. Tactical Visibility Example. Source: Linden (2002).

Linden also showed that navigation graph analysis could support squad tactics such as locating possible ambush nodes (2002). Figure 7 shows this practice in three simulation scenarios involving the exit (or exits) of a room. The first node inside and outside of the room are labeled *I* and *O*, respectively. The pinch node, *N*, is a neighbor of *O* that is on the path to *I*. Nodes that have visibility to *O* but not *N* are possible ambush points. For rooms with multiple exits, multiple pinch points can be identified and covered by other Squad members.

Linden (2002) does acknowledge some limitations of these methods. First, since each node stores information about every other node in the graph, as the graph grows in nodes, memory and computation cost grow exponentially. Also, the effectiveness of the analysis is dependent on properly placed nodes and level design. Linden notes this can be overcome with well-trained developers or better automatic generation of node placement (2002).

that was either static or dynamic. Possible layers included precalculated values such as openness (distance to obstacles) and available cover (Tozour 2004). Other values that could be updated during execution included searched areas, occupancy, and lines-of-fire for other entities in the simulation.

Straatman, van der Sterren, and Beij were able to produce dynamic tactical behavior in their bots through the use of waypoint annotations and position evaluation. In their approach waypoints are annotated based on assigned goals and operating areas. Additionally, dynamic information such as cover, lines-of-fire, and spacing related to other friendly entities are calculated for each waypoint (Straatman, van der Sterren, and Beij 2005). Position evaluation functions are then used to determine scoring for each waypoint and entities pick ones with the highest score. Figure 8 shows their AI bot calculating node values that take account of lines of fire from primary and secondary threats, as well as a preferred fighting range. After totaling all the node scores, the AI selects its desired position. It is important to note that these calculations are done on demand, with the only exception being visibility calculations.

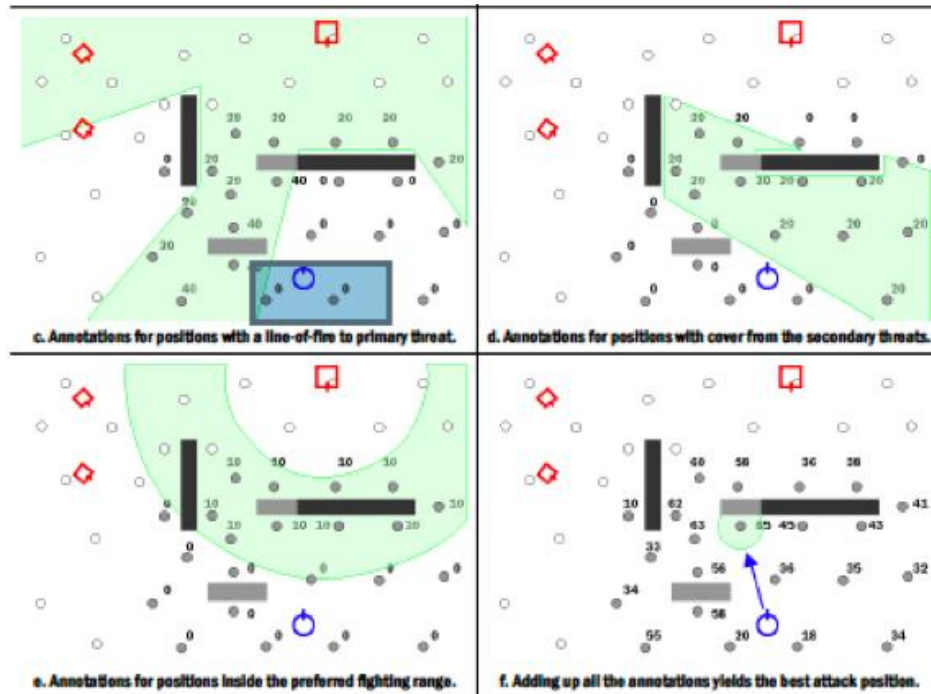
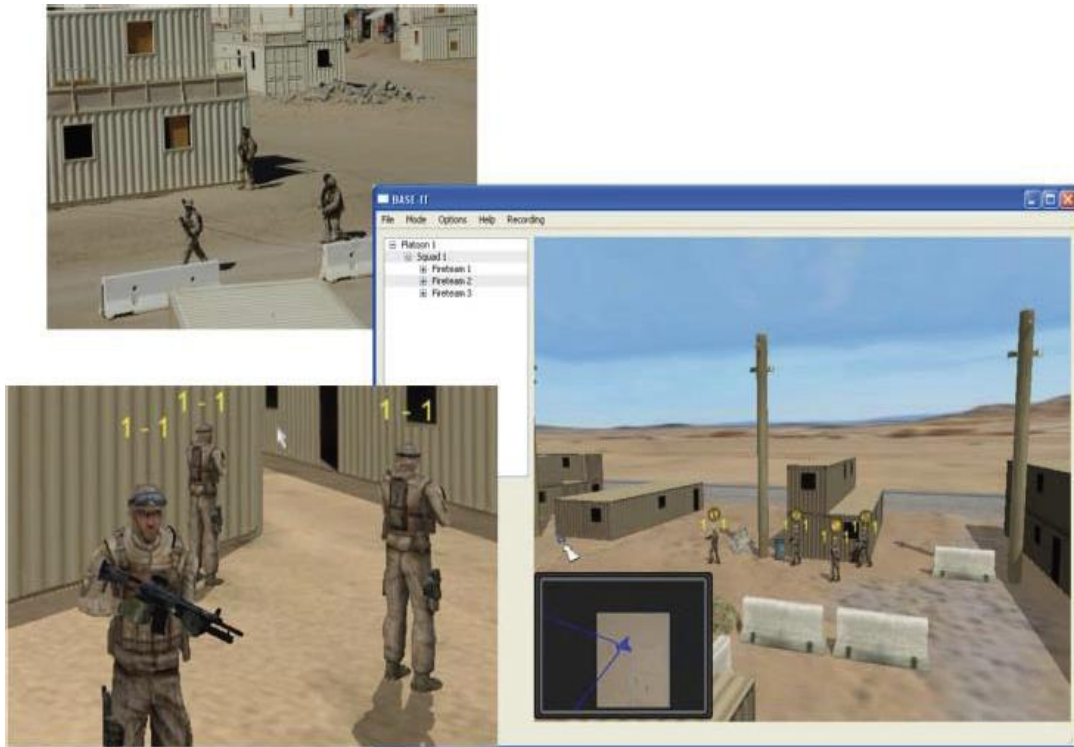


Figure 8. Tactical Waypoint Scoring. Adapted from Straatman et al. (2005).

Research by Darken, McCue, and Guerrero (2010) on fireteam movement also provides some useful equations for the evaluation of squad formations. Looking to emulate the unpredictable, but still focused, manner that fireteams use to move safely through urban areas, they employed several spacing factors in the unit pathfinding algorithm. These include values for maintaining cover, dispersion, cohesion, and distance to their leader (Darken, McCue, and Guerrero 2010). Figure 9 shows an example of real fireteam movement translated into a simulation. While the fireteam may have a determined path, each unit uses these factors to move in their own way. This allows for more dynamic movement that can adapt to different environments than having the unit move in a fixed formation (Darken, McCue, and Guerrero 2010).



Actual Marines in the top image. Examples from the simulation prototype keeping similar spacing in the bottom two images.

Figure 9. Translating Real Fireteam Movements Into a Simulation. Source Darken, McCue, and Guerrero (2010)

4. Prediction in Video Games

In this section, we look at examples of how some of the AI algorithms have been employed in video game environments to predict opposing force positions. Both of the examples use particle filters in addition to training on existing game logs.

Hladky and Bulitko (2008) tested hidden semi-Markov models (HSMM) and particle filter predictors in the first-person shooter, *Counter-Strike: Source*. Their predictor requires that the game map be discretized into a 2D grid. Their HSMM provides the probability of an opponent being in a given cell of the grid for a given time. Their particle filter works similarly to those of Darken and Anderegg (2008), but particles move through the grid according to the HSMM. The HSMM was constructed using player velocity and direction information from the game logs of 140 champion-level Counter-Strike game logs collected from competitive play (Hladky and Bulitko 2008). Their predictor worked well

against tactics that are directly related to player motion, such as rushing to a particular part of the map. However, it had some issues with defensive and standoff tactics (Hladky and Bulitko 2008).

Weber, Mateas, and Jhala (2011) used a particle-based model to improve their automated bot, EISBot, which competed in the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) 2011 StarCraft AI Competition. *StarCraft*, like its sequel (*StarCraft 2*), is an RTS that has been used in AI development. To keep computation costs down, their model only used a single particle for each unit (Weber, Mateas, Jhala 2011). Particle motion was based on the last observed direction, unit type (for velocity), known map choke points, and target destination (Weber, Mateas, Jhala 2011). It is important to note that destination information is not available to human players and could be considered “cheating.” An example of the visual representation of EISBot’s particles in Starcraft is shown in Figure 10. Particles were also assigned a confidence weight that would decay linearly over time. Particles were culled when they either moved to an observable area or had a confidence weight less or equal to zero (Weber, Mateas, Jhala 2011)



The EISBot playing as the green Protoss units. The red squares represent predictions of opponent positions and velocity for units that have moved out of visual range.

Figure 10. The EISBot Playing Starcraft. Source Weber, Mateas, and Jhala (2011).

Interestingly, the EISBot was not concerned with the exact position of each individual unit, but rather the predicted enemy threat of a given region of the game map (Weber et al. 2011). The bot's tactics manager used the region threat values to assign attack locations or allocate idle units to defensive spots on the map. Additionally, EISBot's particle model was interchangeable for different levels of fidelity. This allowed EISBot's performance to be evaluated with the following models:

- Null Model: A particle model that never spawns particles, providing a baseline for worst-case performance.
- Perfect Tracker: A theoretical model that perfectly tracks units that have been previously observed, representing best-case performance.
- Default Model: A model in which particles do not move and do not decay, providing a last known position
- Optimized Model: [A] particle model with weights selected from the optimization process. (Weber et al. 2011).

The evaluation of each model consisted of matches against bots from the 2010 AIIDE StarCraft competition and the game’s own AI system (Weber et al. 2011). Of the four possible models, the optimized model performed the best overall with a win rate of 78% compared to the other models which all had overall win rates of 67% (Weber et al. 2011). EISBot’s creators were surprised to see that the perfect tracker was not the top performer of the three models. They reasoned that with perfect information EISBot did not utilize scouting behavior as the other models did. In addition to providing information about opponent positions, scouting behavior “improved the win rate of the agent by distracting the opponent, such as diverting rush attacks” (Weber et al. 2011). This is another example of perfect information unnaturally and counter intuitively altering the behavior of an AI.

5. Prediction in Military Applications

The Real-time Adversarial Intelligence and Decision-Making (RAID) program conducted by DARPA developed a method to predict opponent locations in urban environments and tested its approach using OneSAF Test Bed (OTB) (Kott, Singh, McEneaney Milks 2011). RAID presents the closest example to what our prediction methods are trying to accomplish. It is able to use sensor data and intelligence reports to generate current and future opponent laydowns. Additionally, RAID uses a Linguistic Geometry approach for COA creation. DARPA compared RAID opponent estimates and COA recommendations against human staff planners in multiple experiments conducted from 2004 to 2006 (Kott et al. 2011). In the experiments RAID estimates outperformed the staff planners in accuracy and battle performance (2011).

6. Summary

In this section, we looked at some of the foundational and current techniques for opponent tracking, position prediction, and the implementation of tactical behavior. The methods usually required some advanced understanding of the simulation environment and opponent characteristics. There was a common need to discretize the environment, through navigation graphs, grids, or regions. Tactical behavior depended on a combination of precalculated visibility, spatial notation of the environment, and dynamic evaluations of

the current situation. We also saw that increases in the number of nodes in the environment, particles, or opponents to track greatly increased computational costs of the methods. The gaming examples we presented both used game logs as training data to develop tactical behavior in their respective AI. While a library of game logs is not always available to a military planner, they do have access to intelligence analysis about the operating environment and enemy tactics. In the next section, we will see how investigations done during the MDMP align with the requirements for tactical position evaluation and prediction.

B. THE MILITARY DECISION-MAKING PROCESS

The MDMP is the process used by military forces to comprehend desired objectives, produce viable COAs, conduct wargaming analysis, and ultimately generate a plan for execution (U.S. Army 2015, 2–1). The intelligence analysis done in the early steps of the MDMP will be utilized by our predictor. Consequently, our predictor can then be utilized for COA creation and wargaming. The main steps of the MDMP are shown below in Figure 11. It is important to note the development, analysis, comparison, and approval of COAs comprise a majority of the steps in the process. Four of the seven steps are directly related to the development, analysis, comparison, and approval of COAs. Furthermore, planners are also required to come up with enemy COAs (ECOA) to support wargaming and evaluation. Good COA develop requires an extensive understanding of the operational environment and opposing forces. Step 2 of the MDMP is where planners develop this understanding.

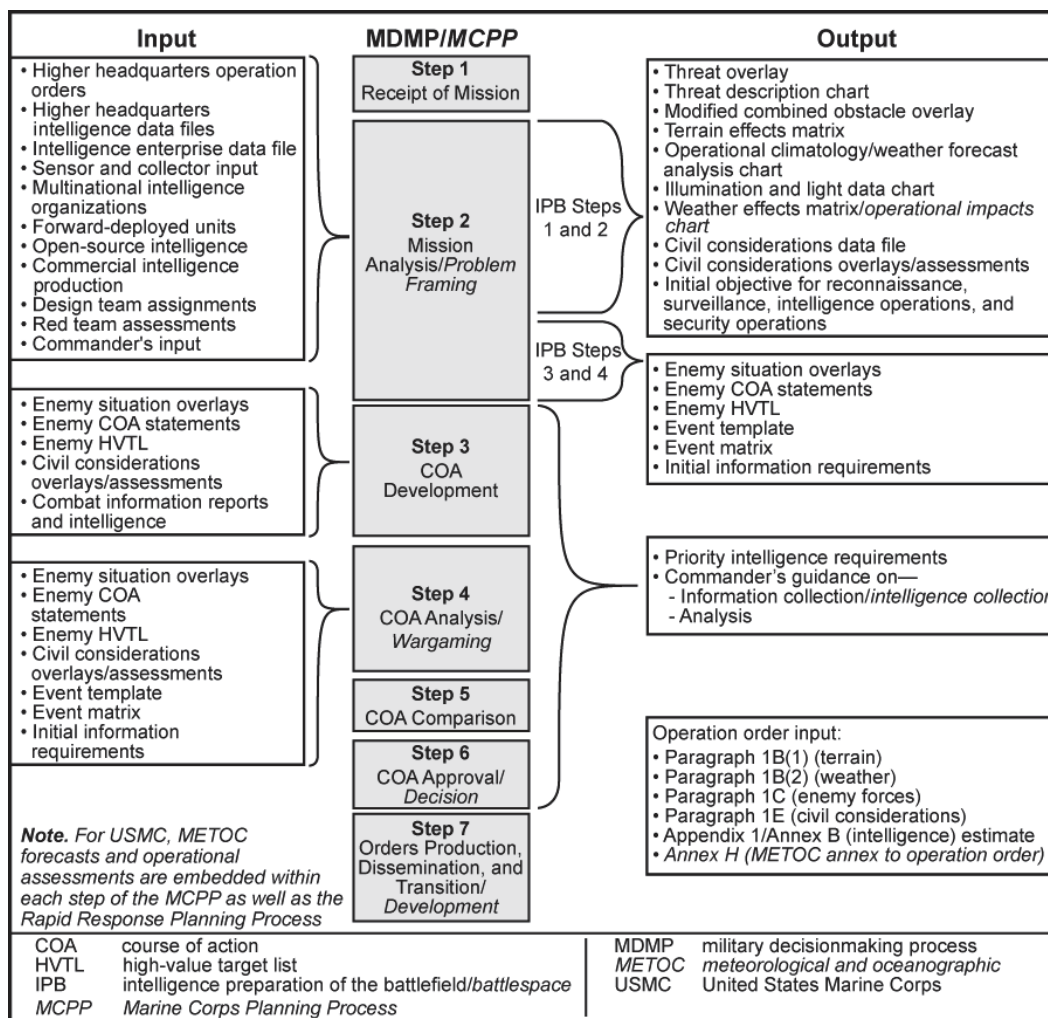


Figure 11. The MDMP. Source: U.S. Army (2015).

In the second step of the MDMP, planners conduct mission analysis. The main part of this analysis is the Intelligence Preparation of the Battlespace (IPB). The IPB is “the systematic continuous process of analyzing the threat and environment in a specific geographic area” (U.S. Army 2015, 1–1). This evaluation includes analyzing available GEOINT and open-source data, as well as studying opposition threat patterns, doctrine, and known standard operating procedures (SOP). The analysis also uses a graphic representation of enemy tactics called an adversary template. The adversary template is a “depiction of what the enemy would do if he followed his doctrine perfectly” (U.S. Marine Corps 2014, 4–6). An example of an adversary template for an ambush is shown in Figure 12.

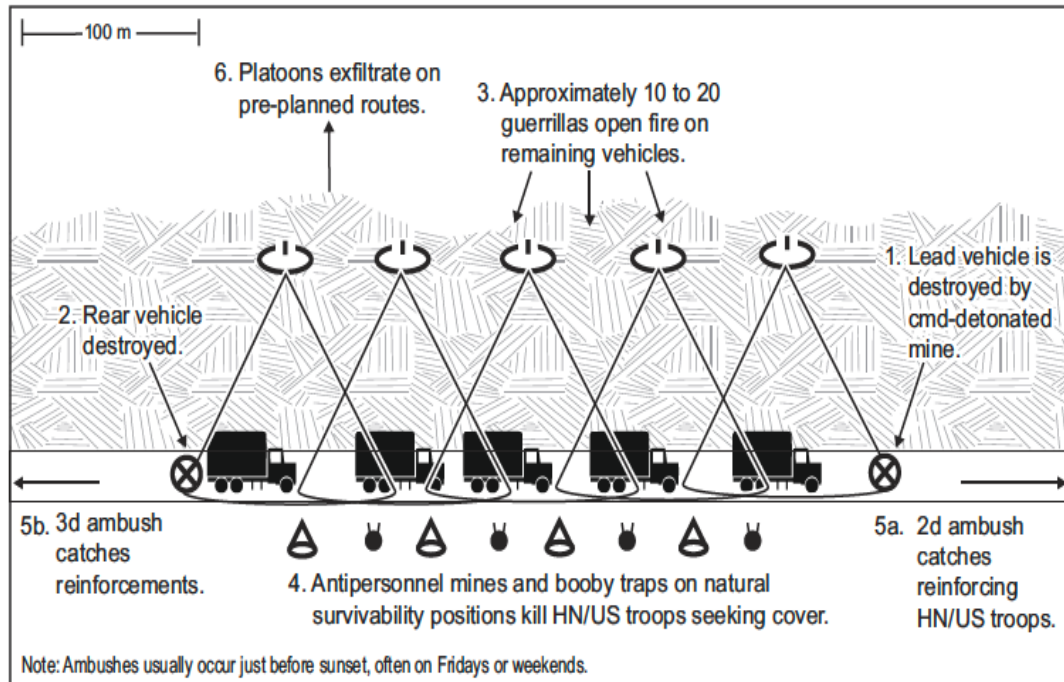


Figure 12. Adversary Template. Source: U.S. Marine (2014).

The adversary template is used with the knowledge of the terrain and weather effects in the battlespace to produce situation templates. The template “predict [s] what the enemy would do if he applied his doctrine to his current place and situation.” (U.S. Marine Corps 2014, 4–7). The situation template can be thought of as an overlay of the enemy SOP with the environment and it is an essential building block of an ECOA (see Figure 13).

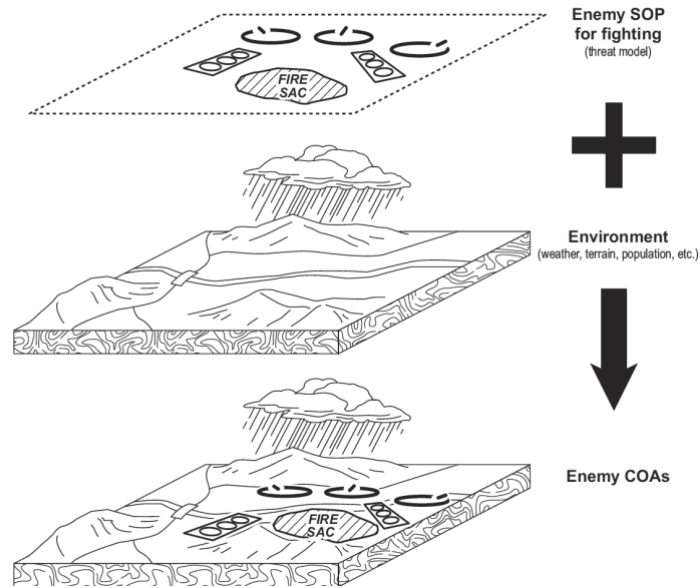


Figure 13. Example of a Situation Template. Source U.S. Marine (2014, 4–9, Figure 4–6).

An ECOA describes expected enemy actions and objectives. Multiple ECOAs are developed to reflect what the enemy may be doing at the start of the operation and during key events (U.S. Marine Corps 2014, 4–8).

These IPB products and analysis can be used to meet some of the requirements of a prediction model and tactical evaluator. GEOINT provides us with our search environment and can be used to determine an effective discretization of the space and/or an appropriate navigation graph. Additionally, elevation and terrain data can be used for precalculated visibility and the identification of choke points. Situation templates could be used to a similar effect as Tozour’s influence maps or spatial databases. Adversary templates provide expected enemy formation geometry and spacing.

In the next section, we will examine a CSE that will allow us to incorporate representations of IPB products into our prediction model.

C. WOMBAT XXI

This section provides an overview of the WOMBAT XXI (WXXI) CSE developed by Harder (2017). His dissertation, particularly Appendix A, provides a more in-depth

description of WXXI and his use of it. WXXI served as the simulation environment that enabled the development, implementation, and evaluation of our methods. Background information is provided here to assist our descriptions of procedures in later chapters.

1. A Simpler Environment for Faster Prototyping

WXXI was created by Harder to support his development of an automated fire support planner. Harder based WXXI on the Combined Arms Analysis Tool for the 21st Century (COMBATXXI) simulation used by the U.S. Army and Marine Corps (Harder 2017). As a means to conduct analysis, COMBATXXI is employed for decision support on such topics as acquisition strategy, future asset development, and tactical uses of forces (Severson 2019, 3). Harder wanted to maintain COMBATXXI’s “stochastic, entity-level, closed-form, discrete-event model of ground combat,”(61) but at “a considerably smaller code base . . . [with] less complex internal and external dependencies” (2017, 351). Previous work by Miller on Hierarchical Task Network (HTN) identified the difficulties of working in COMBATXXI and proposed using the Unity3D game engine as a surrogate development environment (Miller 2016). WXXI is built on the Unity3D game engine and leverages Unity’s “drag-and-drop design, visual debugging capabilities, and the availability of add-on features from the Unity Asset Store” (Harder 2017, 207).

An example of WXXI in the Unity3D user interface is shown in Figure 14. The Unity3D interface provides a clear visual representation of the scene in the top left pane. The top center pane lists the simulation object in a hierarchy and selected object properties are shown in the Inspector pane to the right of the screen. Additional scene elements and scripts can be dragged into the scene from the project pane in the bottom left. The bottom left pane also has a console tab that can provide logging and debugging information to the user. All panes are moveable and can be arranged according to user preferences.

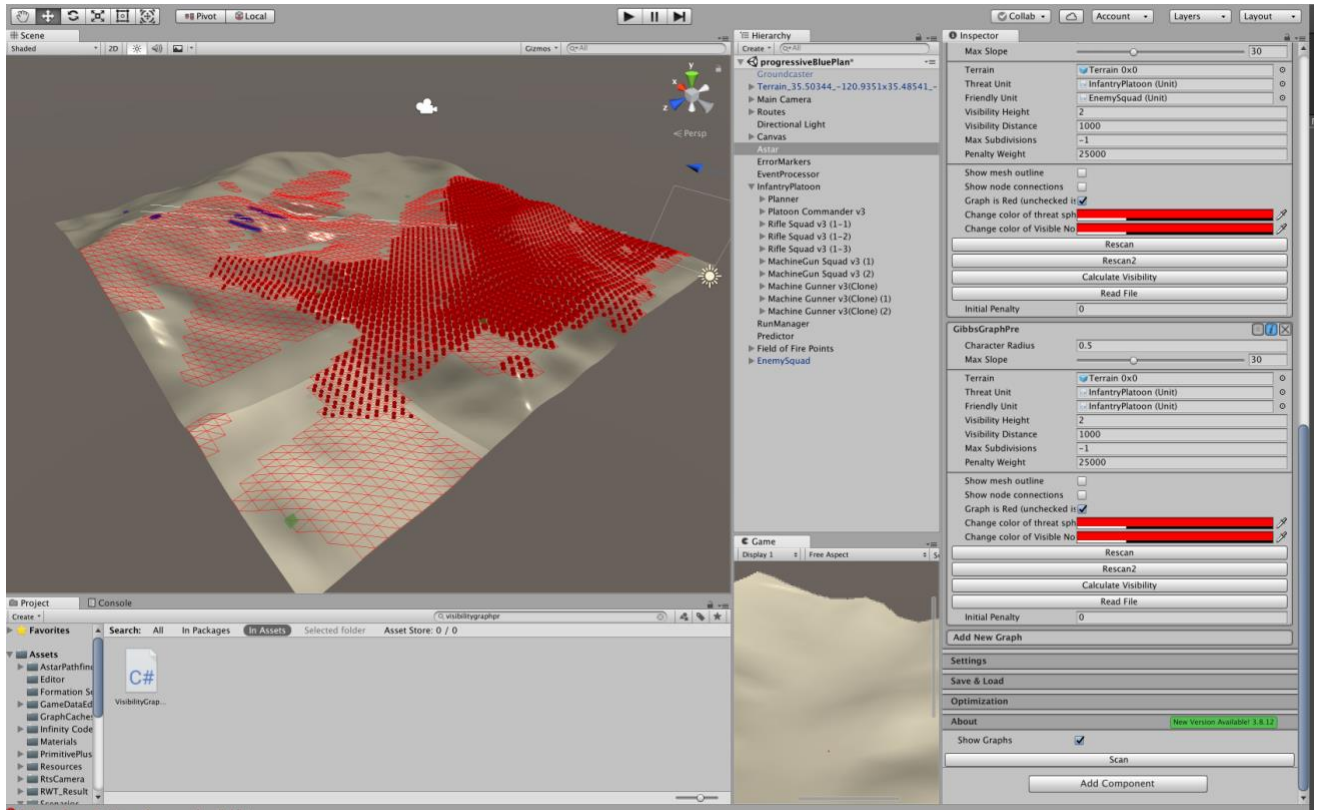


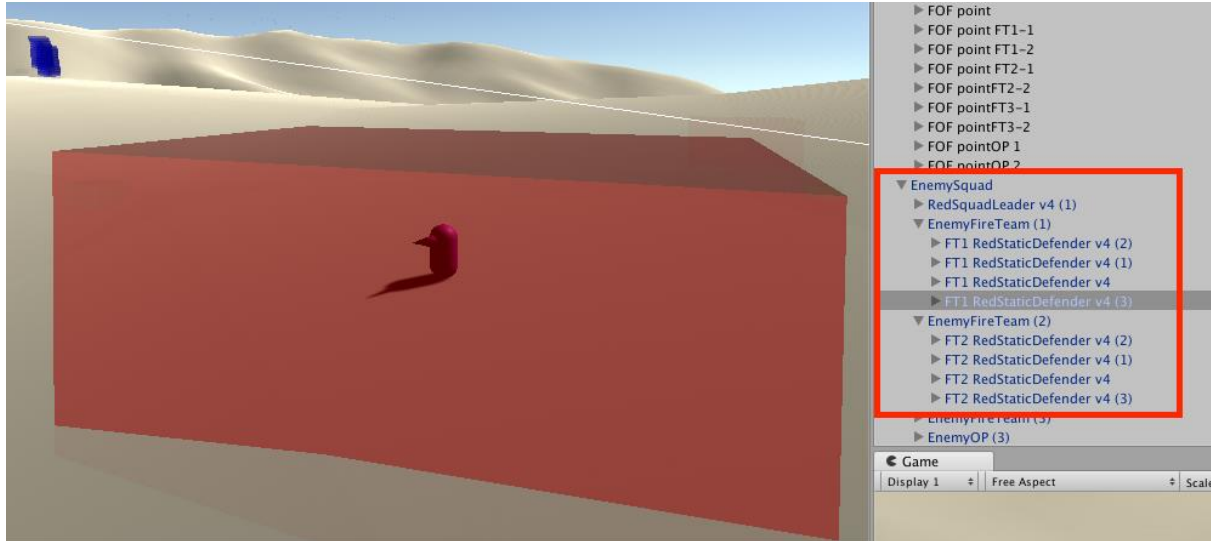
Figure 14. WXXI Implementation in Unity3D.

Relevant aspects of WXXI are discussed in the following subsections. Information on WXXI properties is taken from Harder’s dissertation except where noted otherwise.

2. Entity Representation

Entities represent the basic foot soldier in WXXI and are Unity Game Objects that have assigned properties for movement, targeting, and status. Entities can be given different weapons and are generally assigned to either the red or blue side. Entities are arranged in a hierarchical unit structure similar to COMBATXXI. WXXI takes advantage of Unity’s Transform Hierarchy to make each entity a child of the unit it belongs to. Additionally, each unit can have a commanding unit of which it is a child, or subordinate units that are children of it. Each unit also has an entity that is designated as the unit commander. Figure 15 gives an example of the visualization of an entity and its associated hierarchy. In this example, the entity is a child of its unit, Enemy Fireteam (1). Subsequently, the unit Enemy Fireteam (1) is the child of its commanding unit, the Enemy

Squad. The large red cube around the entity helps with the visualization of the entity as the scale of the simulation is zoomed out.



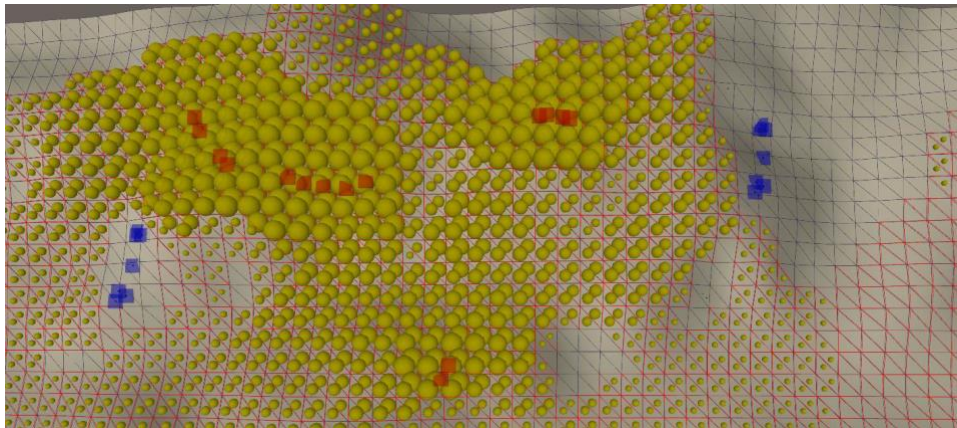
The entity's unit hierarchy is outlined in red.

Figure 15. WXXI Entity and Hierarchy

3. Terrain and Risk Representation

Battlegrounds in WXXI are represented using Unity Terrain objects. These objects are comprised of a triangle mesh with elevation changes represented in the vertices of each triangle. Real-world geospatial data is imported and processed using two packages from the Unity Asset store. First, the Real World Terrain Asset is used to provide elevation data for the Terrain object from a user-defined geographical area. Second, the resulting Terrain object is then processed to create a navigation graph using the Astar Pathfinding Project Asset (Granberg 2016). The navigation graph uses the triangles of the terrain object as the basis for its nodes and is created using custom code written by Harder. He calls the resulting graph a VisibilityGraph. It is important to note that each node in a VisibilityGraph is actually a triangular volume based on the triangles of the mesh of the Terrain object and a user-specified visibility height. A threat Unit is also designated for the VisibilityGraph. Entities that belong to the threat unit are considered threat entities. Each node is annotated by what threat entities have visibility of the node as well as the threatening forces that can

be suppressed from the node. Finally, every node is assigned a risk value. Risk values for each node are used in pathfinding and other calculations in WXXI. The node risk values in WXXI are set according to the expected losses during the traversal of the node. The expected losses calculation is a function of the time to cross the node and the fires that can be put on the node by opposing forces. The risk value increases as more entities have targetability of the node. For any entity, their targetability of a node is the amount of risk that they are able to apply to that node. The targetability of a node is a function of the entity's visibility and range to the node, as well as the entity's weapon type. Generally, targetability increases as the range to the node decreases. Figure 16 shows an example of the terrain node structure and risk annotations. Note that the risk values (indicated by the size of the yellow spheres) increase in the nodes closer to the red force formation.



In this example, both red and blue forces are shown. Additionally, the risk value of a particular node is indicated by a yellow sphere. Larger spheres show a higher risk value.

Figure 16. WXXI Example Showing the Terrain-based Triangle-mesh Navigation Graph. Source Harder (2017).

4. Combat Simulation

Since Unity3D games are generally designed for a time step model with updates according to frame rate, Harder designed his own discrete event system (DES) to handle WXXI combat simulation. His BasicEventProcessor component maintains its own variable for time and an event list on which other WXXI components can schedule actions. It also determines the simulation stop time. Entity movement, sensor recognition, and hit

detection are all handled by the DES. Harder’s implementation ensures that “the outcome of any replication is independent of the graphical frame rate and computational resource competition” (Harder 2017, 353). WXXI leverages one of the key advantages of a DES by allowing battles to happen at any time speed the user desires. WXXI has two modes for execution, Testing and Experimentation. Testing allows for battles to be observed according to Unity Time settings and is useful if the user wants to watch the battle unfold. In Experimentation the graphical representation of the battle is rarely updated. It is useful for collecting data on batch runs of multiple battles. In the WXXI simulation, battle results are stochastic. Further discussion about the visual elements of a battle in WXXI will be covered in Chapter V.

5. Data Collection

In WXXI each battle scenario is represented as a distinct Unity scene. In order to test various modifications to a particular scenario, each modification must be saved as a separate scene. WXXI’s BasicRunManager class will collect data from each battle run and sends it to a comma-separated-value (CSV) text file. In order to facilitate batch testing runs, WXXI includes the MultipleRunManager class. This class can be added to its own Unity scene. It can then be used to run any number of distinct scenarios as separate scenes. Users can specify the scenes to be run and the number of replications of each scenario. Each replication uses a different random number generator seed for the stochastic process in the replication. Data for each replication is collected using each scene’s BasicRunManager and output as a single CSV text file. The specific data elements used in this dissertation will be discussed in Chapter V.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHOD

This chapter presents the framework for building a prediction model based on environmental factors, intelligence products, and observed units. Some of the concepts and algorithms were previously published by Maroon and Darken (2019) but will be discussed here in more depth.

The chapter begins with some of the basic assumptions about a theoretical CSE and the elements in it. The CSE is arranged in a similar fashion as WXXI to allow an easier transition to implementation (as seen in Chapter IV). The methods are not specific to WXXI and would be able to be implemented in other CSEs with some adaptations. We then define the formation scoring function and its components. Next, we show how the scoring function is used to find a predicted formation through hill climbing and describe how a candidate set of predictions is built. Equations relating to the canonical ensemble are utilized on the candidate set to analyze risk and perform basic opponent plan recognition, so we present an overview of the equations and how they are employed.

A. ASSUMPTIONS

In order to make an efficient description of our methods, we will assume that we are operating in a CSE with “friendly” and “enemy” forces present. Friendly entities are represented by f , and the set of all friendly entities is F . Likewise, enemy entities are represented by e , and the set of all enemy entities is E . Within the CSE is a continuous terrain constructed from a triangle mesh. Each triangle in the mesh represents a node n , and the set of all nodes is N . There are no obstructions on the terrain surface and visibility between nodes is established by an unobstructed raycast from the center of one node to another at entity height.

Friendly and enemy entities exist on the surface of the terrain and have the following attributes:

- An entity can observe other entities that are within their visual detection range and located on a node that is visible from the entity’s own node.

- Each entity has a weapon that can apply a threat or targetability value, t , to all nodes that are within the weapon's range and visible from the entity's own node. This threat value represents the expected losses an entity can inflict on an opposing unit as it crosses the node. The threat value increases as the weapon's lethality and rate of fire, improve. It also increases as the entity's distance to a node decreases.
- Entities are arranged in a hierarchical unit structure. Each unit has an assigned unit leader. Units can be comprised of any number of entities or sub-units.

Threat values for each node are distinct and cumulative for each force. That is, each node will have a threat value t_e that is the total threat applied to the node by all entities from the enemy forces and a separate threat value t_f for the total threat applied by the friendly forces. Finally, each side is aware of the order of battle (entity numbers and unit structure) of the other force, but only knows the position of opposing entities that are on nodes that are visible to its own members.

B. FORMATION SCORING FUNCTION

A formation is the mapping of each of a force's entities to the node at which it is located. Each formation will have an associated threat mapping (V) which is the threat value (t_{force}) for each node (n) in N given the position of all of the entities in the formation. In order to make comparisons between distinct force formations, we need to have a way to evaluate how well the formation suits the force's tactics and objectives. A numerical representation of this is the formation score.

While any number of formation scoring approaches can be developed, we use a method that is comprised of assigned target nodes and simple squad geometry. Entities each calculate their own score based on their position. The sum of the position scores of the entities for a force is the formation score. In our method entity position scores are increased for being able to target assigned nodes. Scores are decreased for the entity breaking a desired unit geometric constraint or being in a position that has an associated

penalty. Unit geometry penalties are determined using some of the considerations found in Darken, McCue, and Guerrero's work on realistic fireteam movement (2010). Specific details on the elements of an entity's position score are discussed in the following sections.

1. Preferred Targets and Field of Fire

Each entity can be assigned specific nodes in N that it is responsible for applying threat (t) to. The node assignments are set through the use of preferred targets and fields of fire (FOF). Preferred targets are other entities in the CSE. When assigned a preferred target, an entity will seek to maximize the threat on the node that the target is located on. This is useful if an entity wishes to threaten specific targets in an opposing force or be able to protect designated members of its own force. Entities can also be assigned an area of the terrain to threaten. The set of nodes located within the assigned area comprise the entity's FOF. An entity's positional score is increased by the sum of the threat it applies to each of its assigned nodes. Locations with good visibility of and proximity to assigned nodes will tend to have higher positional scores.

2. Distance to Unit Leader

Entities are expected to stay within a maximum distance to their unit leader in order to be able to receive commands (Darken et al. 2010). Since forces are arranged in a hierarchical structure, entities maintain distance to the leader of the sub-unit they are assigned to. Sub-unit leaders stay within a minimum distance to the leader of the next highest unit in the hierarchy that is directly acting in the current operation. The penalty to an entity's position score for being beyond the maximum distance increases as the distance from the leader increases.

3. Distance to Other Entities

Inter-entity spacing between each other is a balancing act between two requirements. Entities need to be close enough to other unit members in order to provide mutual support. They also need to maintain enough distance between each other to not fall victim to simultaneous attacks. To account for this, Entity position scores are assessed dispersion and cohesion penalties. Units are assigned a maximum and minimum distance

that member entities are expected to keep with each other. Entities that are too close to other members are given a dispersion penalty. Entities that fail to stay within range of the other unit members are given a cohesion penalty. The dispersion penalty increases as entities get closer to each other and the cohesion penalty increases as they get further apart.

4. Lines of Fire

Entities that have been assigned preferred targets or a FOF also have an associated line of fire oriented towards their objectives. Entities will try to stay out of each other's lines of fire. Entities whose location intersects a line of fire will have their positional scores penalized.

5. Cover

It is expected that entities will avoid nodes that have a high threat value from an opposing force. A cover penalty can be assessed in two different ways. First, the penalty can scale according to node's t value from the opposing force. This would give some consideration to range and the number of opposing entities that threaten the node. Second, the penalty can be set at a fixed value for any node that is targetable by any opposing entity.

Another consideration for cover calculations is that it requires knowledge of the opposing force's position, which is not always available. We can work around this by using expected opposing force positions to estimate node threat values. This works reasonably well when we know the opponent's direction of approach or for defined terrain features, such as hilltops.

6. Weighting

Each of the elements in the formation scoring function also includes a weighting multiplier. This enables each element to be scaled according to its importance in a given scenario. Additionally, setting the weighting value to zero essentially "turns off" that element from the scoring function.

7. Scoring Function Algorithm

A simple implementation of a scoring function utilizing the elements described above is shown in Algorithm 1. An earlier version of this algorithm was presented by Maroon and Darken (2019).

Algorithm 1. The Formation Scoring algorithm (n_i : node at position i , t_x : threat value from x , W : weighting multiplier, P : penalty)

```
calculateTotalScore (Unit)
  formationScore = 0
  for all entities in unit
    formationScore += calculateEntityScore (entity)
  for all subunits in unit
    formationScore += calculateTotalScore (subunit)
  return formationScore

calculateEntityScore (entity)
  entityPositionScore = 0
  for all preferredTargets assigned to entity
    entityPositionScore += ( $t_{entity}$ ) at ( $n_{preferredTarget}$ ) * ( $W_{preferredTarget}$ )
  for all FoF n assigned to entity
    entityPositionScore += ( $t_{entity}$ ) at ( $n$ ) * ( $W_{FieldOfFire}$ )
  if(distance to unit leader > max distance to leader)
    entityPositionScore -= ( $P_{distanceToLeader}$ ) * ( $W_{distanceToLeader}$ )
  if(distance to other entities > max cohesion)
    entityPositionScore -= ( $P_{cohesion}$ ) * ( $W_{cohesion}$ )
  if(distance to other entities < min dispersion)
    entityPositionScore -= ( $P_{dispersion}$ ) * ( $W_{dispersion}$ )
  if( $t_{opposingForce}$  at  $n_{entity}$  > 0)
    entityPositionScore -= ( $t_{opposingForce}$ ) at ( $n_{entity}$ ) * ( $W_{cover}$ )
  if( $n_{entity}$  is in a line of fire)
    entityPositionScore -= ( $P_{lineOfFire}$ ) * ( $W_{lineOfFire}$ )
  return entityPositionScore
```

It is important to note that given the hierarchical structure of entities in the CSE a given Force is essentially the unit that all subunits are children of. This allows recursive calls of the **calculateTotalScore** function on subunits to work through the entire Force construction. In this implementation the entity's cover penalty is scaled according to the threat imposed on its node by the opposing force.

8. Building the Scoring Function

The scoring function needs to reflect expected adversary tactics and objectives, as well as environmental considerations. It offers a process to bring real-world intelligence into the CSE. The IPB provides products that can directly populate the scoring function. We can use adversary templates as a means to evaluate unit geometry and annotate graph nodes consistent with enemy objectives outlined in ECOAs. Looking again at the adversary template example in Figure 12, it is easy to see the defined entity spacing, target priorities, and lines of fire, which would all contribute to building a formation score for a suspected enemy ambush. Applying this template to the terrain in our CSE would be similar to the creation of a situation template scene in Figure 13.

C. PREDICTION DEFINITION

We define a prediction as a Force formation that maximizes a given scoring function. More specifically, given a partially observed enemy Force, a prediction represents the positions of the unseen entities, relative to the detected entities, that produce the highest formation score. As each prediction is a formation, all predictions also have an associated threat mapping (V).

Searching for a maximizing set of entity positions is not trivial but can be approached in many ways. Techniques such as hill climbing, simulated annealing, and genetic algorithms can be implemented to find high scoring formations. However, using any of the techniques does not guarantee us to find the exact maximizing set of positions. At best, we can use these algorithms to determine formations that are local maxima in the search space.

D. RANDOM PLACEMENT/HILL CLIMB

While there are a number of approaches to determining entity positions that maximize unit formation score, we employed a method using simple hill climbing from random positions. This process produced consistent results while also allowing for easier customization and debugging.

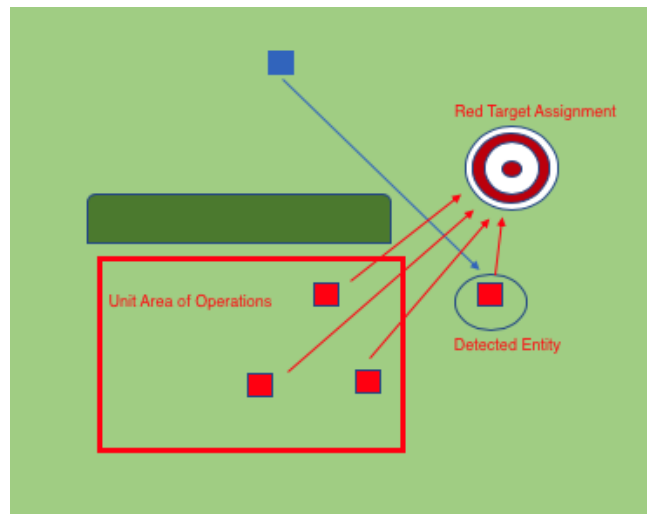
For this example, we will assume that the friendly force (F) is trying to predict the enemy force (E). As discussed in the previous section, detected entities, are not included

in the randomization and hill climbing. The other undetected entities will take positions that maximize the formation score in relation to these detected entities.

It is important to note the process is a manipulation of hypothetical entity positions. The end result of the random positioning and hill climbing process represents a potential formation that unit could be in, according to the formation scoring function.

1. Randomization

When selecting random positions for each entity, we restrict our selection to spots that are within a given area defined by the expected location of the entity's unit. We call this selection region a unit's area of operations. Even though the friendly force does not have exact positions for the undetected entities, it is reasonable to assume that they have identified probable areas for each enemy unit. In building the formation scoring function, the friendly force would have identified Enemy targeting priorities and spacing. They would expect that a unit's area of operation would be located near their targeting assignments and detected unit members. Figure 17 gives an example of an area of operation for a red unit hidden behind an obstruction. Larger areas of operations generally reflect less certainty about where the unit's entities could be.



Given the detected red entity and known targeting assignment, the red unit's area of operations is shown as the red rectangle behind the obstruction.

Figure 17. Area of Operation Example.

2. Hill Climbing

Upon being placed on its new node, the entity calculates its position score. Then it identifies its node's neighbors and evaluates what its position score would be at these nodes. The entity then moves to the node with the highest position score that is also unobserved by Friendly Forces. This process continues until the entity has moved to a node with a higher position score than all of its neighbors. When all undetected entities have moved to their maximizing locations, we have identified a possible maximizing formation.

3. Hill Climbing Algorithm

The hill climbing method is shown in Algorithm 2. An earlier version of this algorithm was presented by Maroon and Darken (2019).

Algorithm 2. The hill climbing algorithm (n_i : node at position i ,)

```
hillClimb (unit)
  for all entities in unit
    if (entity is detected)
      continue

    currentNode =  $n_{entity}$ 
    highestScoringNode = currentNode
    higherNodeIsAvailable = true

    while(higherNodeIsAvailable)
      for all neighborNodes to currentNode
        calculateEntityScore(entity) at currentNode
        calculateEntityScore(entity) at neighborNode
        if (Score at neighborNode > Score at highestScoringNode)
          highestScoringNode = neighborNode

      if (currentNode = highestScoringNode)
        higherNodeIsAvailable = false
      else
        moveEntityToNode(highestScoringNode)
        currentNode = highestScoringNode
        higherNodeIsAvailable = true

  for all subunits in unit
    hillClimb (subunit)
```

Similar to the formation scoring algorithm, the hill climbing algorithm uses recursion to work through the hierarchical Force structure.

E. HANDLING LOCAL MAXIMA

A potential issue with using hill climbing to maximize the scoring function is that we cannot be assured to find the exact highest-scoring formation. The algorithm is capable of identifying formations that are the local maximum based on the random starting positions of the undetected entities, but even with a simple formation scoring function, the search space can be expected to have multiple peaks. One possible solution is to repeat the process multiple times, with new random starting positions for each repetition. Each iteration produces a distinct prediction that can be further evaluated according to its formation score.

The simplest representation of the likely enemy positions will have the highest scoring formation designated as the prediction. However, it would be reasonable to assume that human commanders would have an approximation of the range of possible locations where their opposition could be, in addition to their exact prediction. It is possible to model this rough estimate using the candidate set of predictions generated through multiple iterations of random repositioning and hill climbing.

Within the candidate set, there exists any number of acceptable formations that have scores that are close to the highest score. We can determine the acceptable formations by establishing a probability value between 0 and 1 for each formation in the candidate set. The sum of all the formation probability values will be equal to 1. Formation probability increases with higher formation scores. Possible formations can then be selected from the candidate set according to their respective probability value.

We can also use the candidate set probabilities to reflect the assumed accuracy of the scoring function. This is accomplished by giving the candidate set a certainty value that can influence each of the formation probabilities. As the certainty value decreases, all formations in the candidate set become equally acceptable, regardless of the formation score. As it increases, the higher scoring function becomes more probable, while lower

scoring formations approach zero probability. At the highest levels of certainty, only a few of the highest-scoring formations are considered.

The next sections show how we develop a method to sample from the candidate set at a given level of certainty using a Gibbs distribution.

F. CANONICAL ENSEMBLE

This section provides background material on some of the statistical mechanic concepts and formulas used in developing our Gibbs distribution. Most of the details are adapted from Glyde’s presentations for his courses at the University of Delaware (2010).

Statistical mechanics provides the means to understand and model the states of large collections of atoms. Glyde identifies two key purposes for statistical mechanics. First, it is useful to determine the distribution of identical systems across possible states (2010, 1.1). Secondly, it “provides the link between the microscopic properties of a many body system and its macroscopic character” (Glyde 2010, 1.1). Ludwig Boltzmann and Willard Gibbs provide most of the foundational work in statistical mechanics. For our purposes, we will be focusing more on Gibb’s work with ensembles that Glyde describes as

[The] system [being studied] may, however, be in a number of states. We naturally want to include and describe all possible states of the system. To do this we construct an assembly of mental copies of the system in its different states. . . each possible state must appear at least once in the assembly. (Glyde 2010, 1.3)

Thermodynamic systems are described in terms of their volume, number of particles, temperature, energy, and entropy. Each mental copy of the system represents a possible state that the system could be in. A canonical ensemble is “an assembly of mental copies” of the possible states of a thermodynamic system in a heat bath with a constant temperature (Glyde 2010, 6.2). The system itself has a constant volume, number of particles, and because of the heat bath, temperature. However, energy is not constant because it can flow (as heat) between the system and the heat bath. Thus, each of the mental copies can have a different energy state (E_j). The canonical ensemble has several useful properties shown in the following sections.

1. State Probability

To find the probability (P_s) that one of the mental copies is the actual state that the system is in, we use the equation:

$$P_s = \frac{1}{Z} \exp^{-\beta E_s}$$

where β is the Boltzmann factor for a given constant (k) and temperature (T). It is given by the formula:

$$\beta = (kT)^{-1}$$

2. Partition Function

In the state probability equation above, Z is the sum of all the possible states better known as the partition function:

$$Z = \sum_s \exp^{-\beta E_s}$$

3. Energy

The total energy (U) of the system given all of the possible energy states is the sum of all the states multiplied by their probability:

$$U = \sum_s P_s E_s$$

4. Entropy

The entropy (S) of the system can be calculated by the following equation:

$$S = -k \sum_s P_s \log P_s$$

When selecting from the possible states, those with lower energy are more probable than those with higher energy. Additionally, entropy increases as temperature increases,

and as it rises, all states become equally probable. In the next section, we use these equations to form the basis of our implementation of the Gibbs Distribution.

G. GIBBS DISTRIBUTION

Our set of candidate predictions has properties similar to those of a canonical ensemble. The candidate prediction set acts as our ensemble, with each formation representing a possible state that the system could be in. The next section applies some of the previous formulas to our specific search space.

1. Formation Probability

Given j rounds of hill climbing, with each round producing a distinct prediction formation, we can calculate the probability the actual formation (X) is one of the prediction formations (x) using the following:

$$P(X = x) = \frac{1}{Z(T)} \exp \left(\frac{\psi(x)}{T} \right)$$

where $\psi(x)$ is the formation score given x , and $Z(T)$ is the partition function:

$$Z(T) = \sum_{i=1}^j \exp \left(\frac{\psi(x_i)}{T} \right)$$

The value T represents temperature and is determined by the desired entropy of the distribution. Usually in statistical mechanics, the value k (seen in the equations in the previous sections) equals Boltzmann's constant. For our distribution, we set the value of k to equal one. As mentioned earlier, the probability for each formation rises as the formation score increases.

2. Distribution Entropy

We can calculate entropy (S) of the distribution with the following:

$$S = - \sum_{i=1}^j P(x_i) \log (P(x_i))$$

where $P(x_i)$ is the probability for each formation in the candidate set. We normalize the entropy so that is a value between 0 and 1:

$$\tilde{S} = \frac{S}{\log(j)}$$

The entropy calculation is then used to set the temperature for a given entropy value, allowing the model to reflect different levels of uncertainty. As the entropy approaches 1, all candidate formations become equally probable, reflecting a greater uncertainty.

This section demonstrated a method to calculate probabilities for formations in the candidate predictions set according to the Gibbs canonical ensemble. In the next section, we will demonstrate the further application of the candidate prediction set to risk evaluation.

3. Aggregated Risk Mapping

The candidate formation set produced by multiple rounds of hill climbing can also be used to calculate a threat value (t) for each node (n) that aggregates each of the threat maps (V) from all possible formations. As stated earlier, each node has a value t_{force} which is the total threat applied to the node for a given force formation. This calculation requires perfect knowledge of threatening force's position. If we do not know exactly where the opposing force was located, we would need a way to calculate threat values according to the probabilities of the formations in the candidate set.

For a given terrain the number and location of nodes will remain constant, regardless of any changes in formation. Recall that in the canonical ensemble, the energy of the system could be calculated as the sum of all the possible energy states multiplied by their probability. We can make similar calculations with our node threat values, given the candidate formation set.

Given a candidate set with j number of formations, we will have j different threat maps (V), one for each formation. Additionally, since each formation has an associated probability, we can also assign a probability to each threat map. We can then create an

aggregated uncertainty threat map (U) with the same node structure as the other threat maps in the candidate set. The assigned threat value for each node, $n(\text{aggregated})$, in U is the sum of the node's threat in each threat map, $V(n(t))$, given its formation's probability ($P(x)$), or:

$$n(t_{\text{aggregated}}) = \sum_{i=1}^j V_i(n(t)) * P(x_i)$$

Calculating risk in this way tries to capture the manner in which human commanders assess risk across multiple possible opposition formations. While the commander may be uncertain of the exact position of the enemy, they will generally have an idea of the most probable threat position and act accordingly. This method is similar to that employed by the EISBot (Weber 2011), in that the abstracted risk of the opponent is used, rather than an exact position prediction. Further, we can model higher levels of uncertainty by using distributions with a higher entropy.

H. PLAN RECOGNITION

As discussed in the previous section, the scoring function is built according to a specified ECOA. The IPB process will develop multiple possible ECOAs that could be valid for a particular battlespace. Each of these ECOAs would in turn generate its own scoring function. Since the scoring function is a vital piece of our prediction method, we would like to ensure that we are using the correct one.

A possible approach to formation score function selection makes slight modifications to some of our previous methods. The Gibbs equations we use to make evaluations of the formation candidate assume multiple possible formations for a single given formation scoring function. We could just as easily evaluate a set of possible scoring functions for a known formation. Just as we were able to assign a likelihood for a predicted formation being the actual formation, we can assign a probability that an opposing force is employing a particular COA.

The main modification made to enable scoring function evaluation is that only detected opposition entities are considered when calculating formation score. Instead of a candidate set of distinct formations, we will have a candidate set of functions each

evaluated against the known opposition entities. We would expect that the more the observed opposition entities aligned with a particular proposed ECOA the more likely it is the actual plan that they are employing. Similarly, the higher the formation score a function generates for known entities, the higher the probability it is the correct function to use.

Given k possible formation scoring functions (ψ), we can calculate the probability one of the functions is the actual scoring function (Ψ) using the following:

$$P(\Psi = \psi) = \frac{1}{Z(T)} \exp\left(\frac{\psi(d)}{T}\right)$$

where $\psi(d)$ is the formation score given the detected entities, and $Z(T)$ is the modified partition function:

$$Z(T) = \sum_{i=1}^k \exp\left(\frac{\psi_i(d)}{T}\right)$$

As in the evaluation of the candidate formation set T is a temperature value that can be used to set a desired entropy value. Entropy is again used to represent a level of uncertainty. Additionally, each scoring function can be used to generate its own uncertainty threat map that is an abstracted risk representation of its candidate set of formations. It is possible to calculate an aggregated risk across all possible scoring functions using their uncertainty threat maps (U) and associated probabilities:

$$n(t_{plans}) = \sum_{i=1}^k U_i \left(n(t_{aggregated}) \right) * P(\psi_i)$$

I. SUMMARY

In this chapter, we presented theories and techniques we used in building our predictor. We demonstrated how outputs from the IPB can be used to craft a scoring function for formation evaluation. We outlined our procedure to populate a candidate set of formations using the local maxima identified in rounds of hill climbing. We provided background on Gibb's canonical ensemble and showed its similarities to our candidate set of formations. We then adapted equations from the canonical ensemble to evaluate the

formations and aggregate risk accordingly. Finally, we presented a way to evaluate potential ECOAs according to their formation score. In the next chapter, we will demonstrate how these methods were implemented in WXXI.

IV. IMPLEMENTATION

In this chapter, we show how the methods discussed in Chapter III were implemented in the WXXI CSE. We initially discuss configurations and additions to WXXI components. This includes modifications to the VisibilityGraph and the new Predictor Class. Next, we provide examples of the methods operating in WXXI. For each of the methods, we examine implementation considerations and any difficulties encountered in their development.

A. CONFIGURING AND MODIFYING WXXI

The development and testing of methods were accomplished in WXXI's Unity Editor without the use of the combat simulator. This allowed us to take advantage of the modular development offered by Unity. The bulk of our methods are developed in new classes added into WXXI scenes. Modifications to existing WXXI classes were kept to a minimum in order to avoid changes to the combat simulator's functionality.

The following sections detail important WXXI components that were used or modified to support the implementation of our methods.

1. Terrain

For our development, testing, and evaluation of prediction methods we used a terrain object created by Harder based on a 2 km² area from Cayucos Creek, CA, shown in Figure 18. The Cayucos terrain is made of 8192 triangles, with the edge of each triangle about 31 meters long (Harder 2017). Each of the triangles also represents a distinct node. This terrain was useful because it offered a variety of hills and plains that enabled testing various aspects of the proposed algorithm.

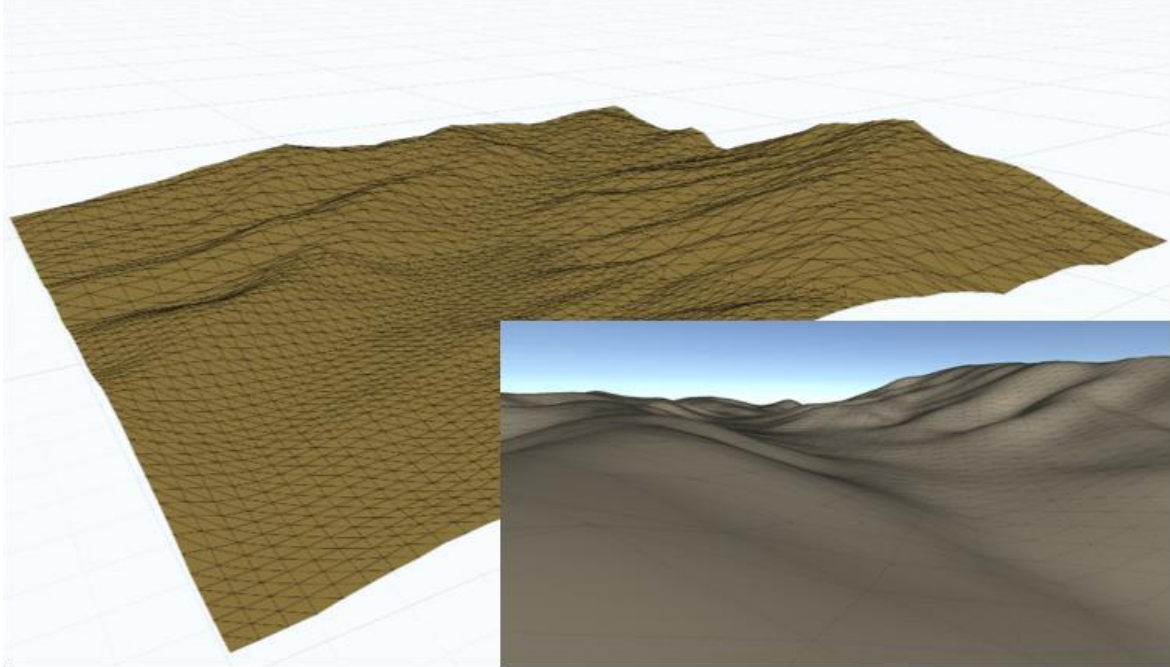


Figure 18. The Cayucos Creek Terrain. Source Harder (2017).

2. Units and Entities

The force structures for our development scenarios included an attacking blue platoon and a defending red enemy squad. We assumed that blue would be trying to make predictions about red's defensive positions. Since red was defending, their positions were considered to be static.

With regard to units and entities, we added the following features.

a. Randomization of Unit Member Positions

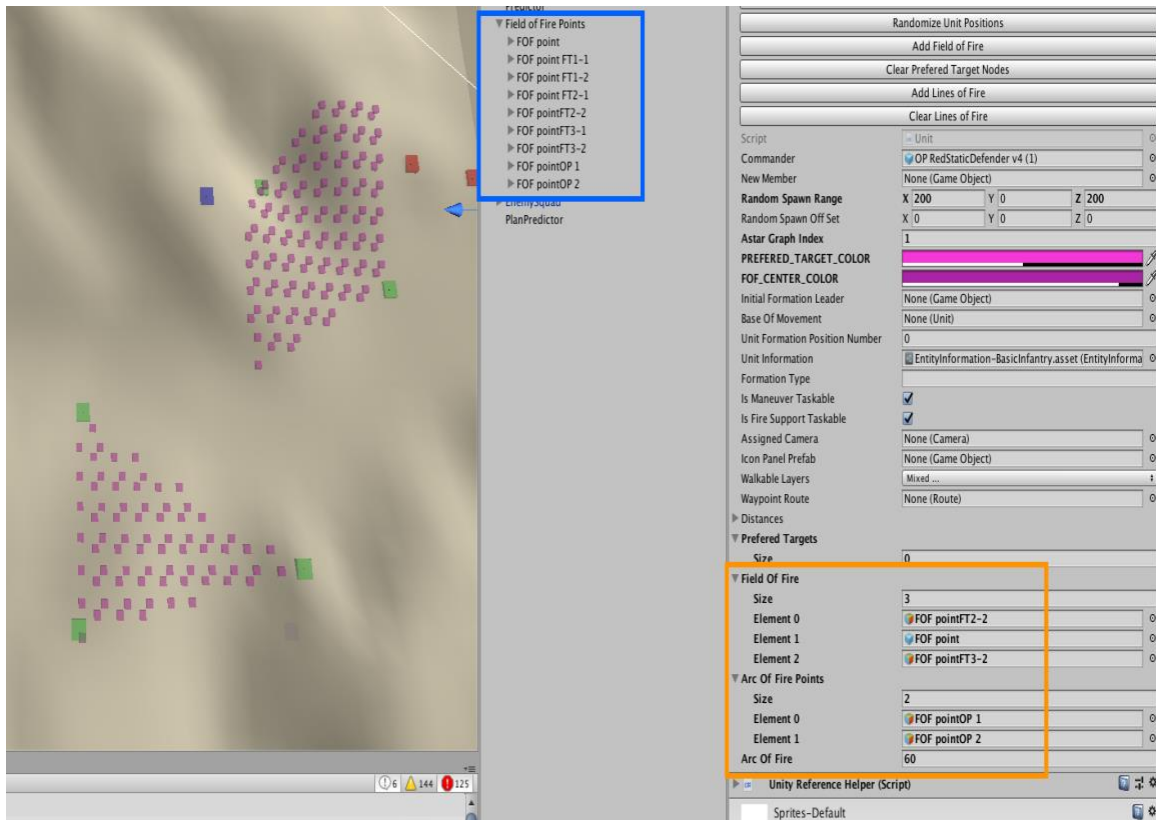
The random placement of entity positions is handled by the entity's unit. Units have a Vector3 random spawn range. This is how the area of operation is defined for the unit. The Vector3 defines a box on the terrain where the Unit's entities can be positioned. The random spawn range is used with the unit's Unity GameObject transform to generate a random position for each entity in the unit and all of its subunits. The entities will then move to the node that is closest to the random position.

b. Assignment of Preferred Targets and FOF

Entity preferred targets and FOF are assigned by the entity's unit. These assignments only affect the entities that are directly a member of a particular unit. Targeting assignments are not passed to entities in a unit's subunits.

Preferred targets are assigned as a designated GameObject. Entities will target the node that the GameObject is on or closest too.

A FOF can be designated by two methods. First, a FOF can be designated by a triangle formed by the node position of three GameObjects. Second, a FOF can be set by a circular sector created by the node position of two GameObjects and an angle value. All nodes in a particular FOF will have an assigned color to aid visualization. An example of the two different methods for FOF designation is shown in Figure 19. In constructing a WXXI scene, we found that it was best to keep a separate group of GameObjects for FOF points (seen in the blue square in Figure 19). These GameObjects would be named according to their associated Unit assignments to make sure shifting a unit's FOF was done easily and correctly.



The FOF points are shown as green squares in the scene. The designated FOF nodes are shown in purple. The blue square in the Editor shows a grouping of FOF GameObjects. The orange square shows the associated FOF GameObjects to the Unit target assignments.

Figure 19. FOF Designation.

c. *Lines of Fire*

An entity's line of fire is determined by target location, its equipped weapon, and a fire angle value added to its Target Handler. Entities have a line of fire for each assigned preferred target and FOF. The line of fire will cover the nodes in a circular sector given by the entity's position and fire angle, out to its weapon range. The direction of the line of fire depends on the type of target. If the type is a preferred target, the lines of fire will have the center of its circular sector arc oriented on the target. For a FOF designated by a triangle, the center will be oriented to the center of the triangle. For a FOF designated by a circular sector, the line of fire will be oriented to the center of the FOF sector arc. Figure 20 shows an entity with three different kinds of line of fire designations.

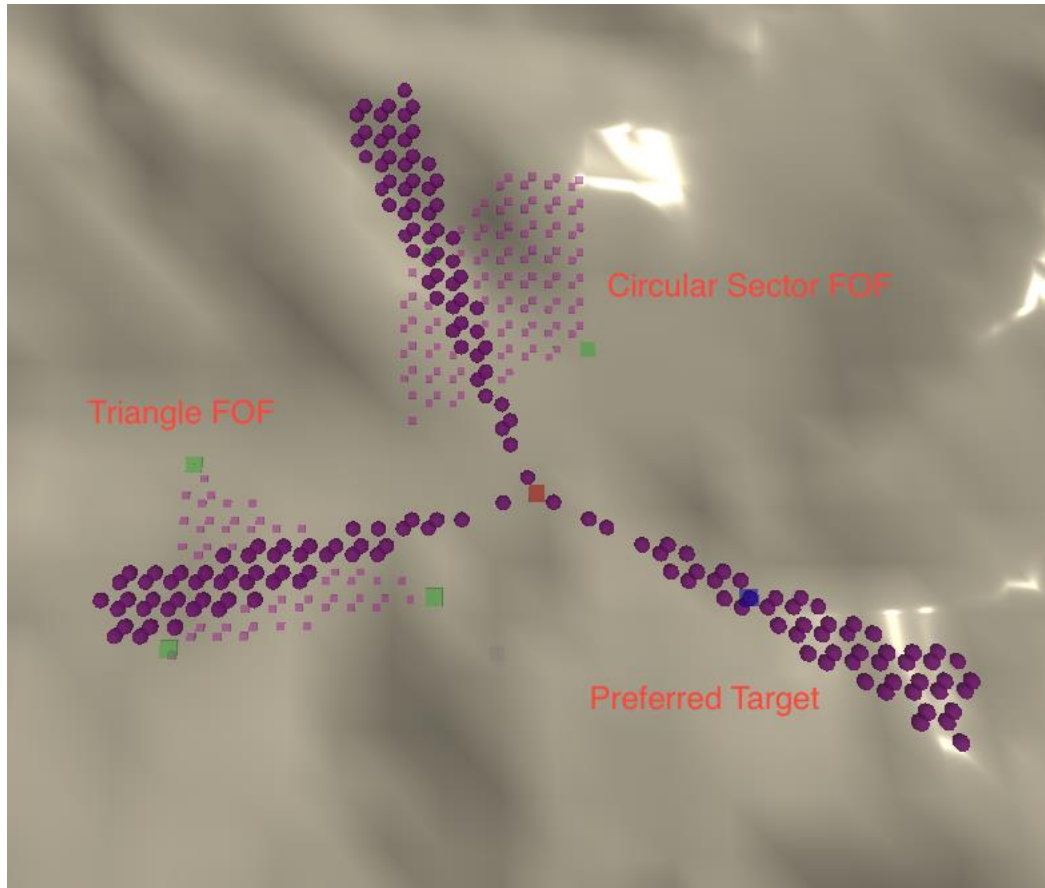


Figure 20. Line of Fire Designation.

3. VisibilityGraph and VisibilityGraphPre

As mentioned in Chapter II, the VisibilityGraph contains the node risk values and annotations for threatening and targetable units. For Harder's fire support planner, a single VisibilityGraph, with red as the threatening force, has to be generated at the start of the simulation. This is done with a call to the ScanInternal() function that is part of the Astar Pathfinding Project package (Granberg 2016). This function handles the creation of node data from the associated terrain and all of the visibility checking and threat calculations necessary for node annotation. Risk values and annotations directly reflect the exact position of red units. Since the red force is static in the simulation, once a VisibilityGraph is created it does not need to be updated.

While Harder's fire support planner only needs the one-time creation and annotation of a single VisibilityGraph, our prediction methods require multiple graphs with frequent updates. Every movement of red forces, whether for randomization or hill climbing steps necessitates a recalculation of risk values and new annotations on the VisibilityGraph. Additionally, we require a separate VisibilityGraph that calculates risk values based on blue being the threatening force. This graph is used for cover calculations in the scoring function. Finally, we need to be able to give the VisibilityGraph risk values and annotations that do not reflect the actual positions of the threatening force.

Given the added complexity of our requirements, multiple calls to the ScanInternal() function become cumbersome, for two reasons. First, once the node structure has been created by the initial ScanInternal() call, we do not need to rescan the terrain and rebuild the nodes from scratch for subsequent updates. Second, the annotation and risk calculations depend on visibility testing using raycasting. This can be expensive for performance given the number of nodes that must be checked for each entity. These difficulties caused us to develop a simpler method for updating node information.

As mentioned early, we tried to minimize changes to existing WXXI classes to ensure proper functioning of the combat simulator. Since we were looking at making significant changes to how the nodes were annotated and updated by the VisibilityGraph class, we decided to use a separate class called VisibilityGraphPre for development and implementation. VisibilityGraphPre is identical to VisibilityGraph, but it features a simpler rescan() function and uses precomputed visibility for its risk and annotation updates. Note we will still refer to the node structure created and updated by the VisibilityGraphPre class as a VisibilityGraph.

The VisibilityGraphPre.rescan() function call is similar to the ScanInternal() call, but it does not perform the terrain scanning and node building. VisibilityGraphPre still requires an initial call to ScanInternal(), but subsequent updates to the created VisibilityGraph can be handled by the rescan() function. The rescan() function updates nodes similar to ScanInternal() but it uses a lookup table to determine visibility between nodes. VisibilityGraphPre can perform a one-time mapping of visibility between all nodes and save the results to a text file. This mapping is read into an array when

VisibilityGraphPre is initialized. All visibility checks in VisibilityGraphPre are then handled by a simple look up rather than using raycasting.

One change that was made directly to the VisibilityGraph class was the added ability to use alternate risk values for nodes. This change had to be made to the VisibilityGraph class directly since we wanted to be able to provide the combat simulation VisibilityGraphs that could reflect the fog-of-war. When using alternate risk values, node risk is read in from precalculated values in a text file. Node annotations are determined using the abstracted unit positions rather than the exact entity positions. We could not completely eliminate the annotations as this would break Harder's fire support planner, which we needed for evaluation purposes (seen in Chapter V).

4. Predictor Class

Our main addition to WXXI is the Predictor class where the majority of the methods from Chapter III are implemented. The Predictor is attached to a Unity GameObject in the WXXI scene and it gives the user access to a custom Editor in the Unity Inspector. A portion of the Editor is shown in Figure 21 and a more detailed description is provided in Appendix A. The Editor is the main user interface for building the scoring function, performing hill climbing, and constructing the candidate set of formations. The Predictor class also handles the initialization and serialization of VisibilityGraphs, Force Positions, and Gibbs distribution results. Initialization includes reading in files for precalculated visibility and setting targeting assignments. Serialization is how results from the Predictor class are output to text files for repeated use. When a battle simulation is run in WXXI, not all of the elements present in the Unity Inspector window are retained. This includes components such as candidate sets and scoring results. Serialization allows us to use a single data set across multiple simulator runs.

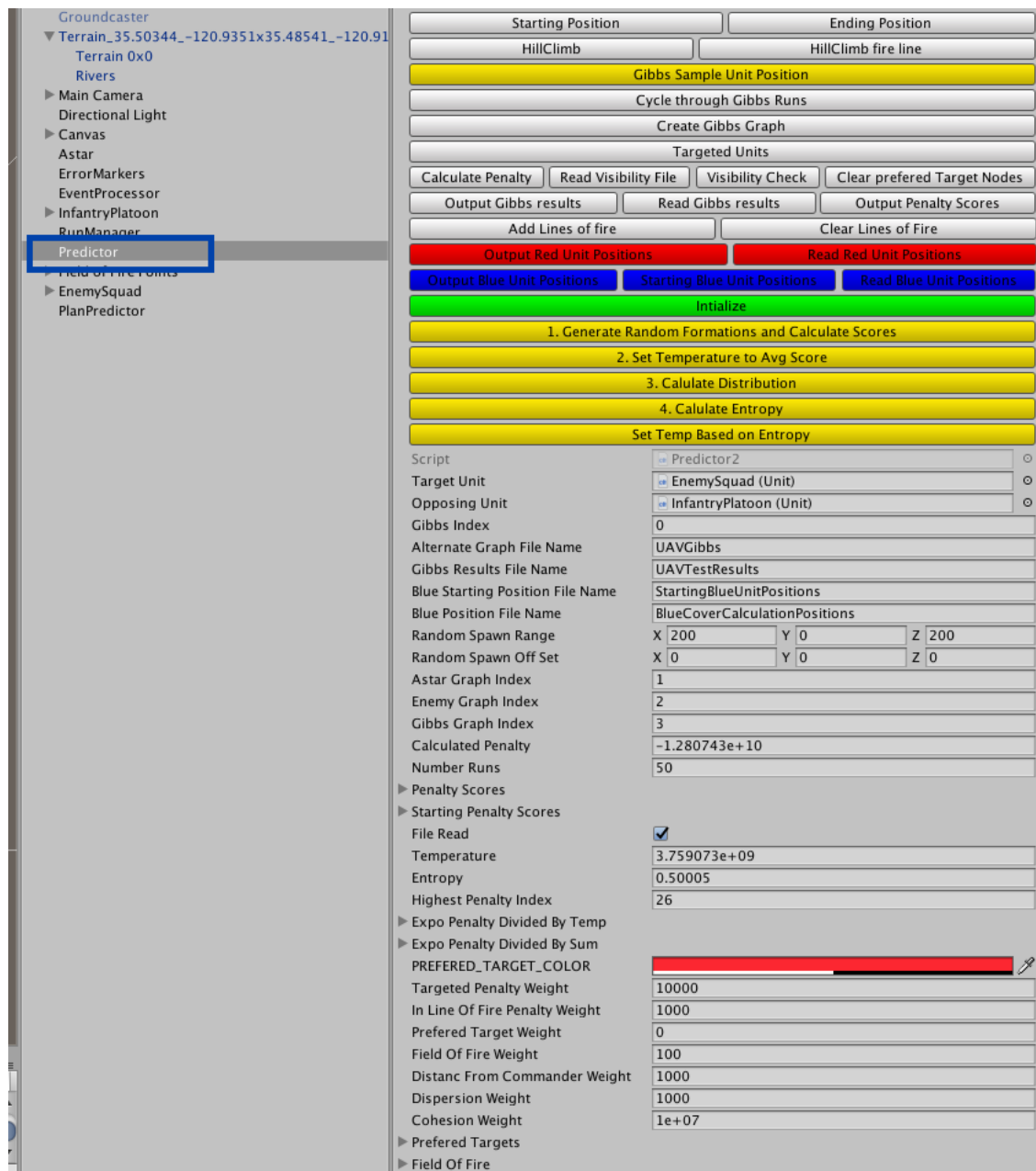


Figure 21. The Predictor GameObject (Blue Box) and Editor.

Specific Predictor class functionality will be presented in each of the method implementation sections of this chapter.

B. FORMATION SCORING FUNCTION

The formation scoring function is implemented by the Predictor class consistent with Algorithm 1 in Chapter III. Individual scores are calculated for each entity in the formation and then they are all summed together for a total score. Specific calculations for scoring function components are described in the following sections

1. Preferred Target and FOF

As discussed previously, entities are assigned preferred targets and FOF at the unit level. This gives the entity a set of nodes that it is responsible to threaten. This threat calculation is similar to the risk value used by the VisibilityGraph to set the risk for each node in the graph. It is a representation of the expected kills the entity will inflict on the opposing unit as it crosses the node. Since Harder's fire-support planner deals with an attacking force he considers this penalty value "risk" (2017). As we are dealing with the defending force, we see this value as "threat."

While the terminology we use is different, the calculation is made in the same way. VisibilityGraphs are given a threatening unit and a friendly unit. The friendly unit is a representative unit for the expected kills determination. The time it would take the friendly unit to move across the longest part of a node is the duration. The friendly unit also has a protection value and a platform type. Additionally, each entity has a weapon component and a targetHandler component.

The Predictors' calculatePenaltyOnNodeFromMember() function is a more lightweight implementation of the node threat calculation done by VisibilityGraph's ScanInternal() function. CalculatePenaltyOnNodeFromMember() uses the entity's targetHandler component to calculate threat with the RiskIntervalValue() function. The RiskIntervalValue() function uses the entity's weapon component to calculate a probability of kill against the friendly unit, given the unit's protection value and platform type. The node threat value is then calculated as the probability of kill multiplied by the weapon rate of fire and the friendly unit's duration on the node. CalculatePenaltyOnNodeFromMember() is called for every node that is assigned to an entity either through a preferred target or FOF.

It is important to note that node threat values are usually calculated to a rather large value. Harder explains that,

Since risk [threat] is measured in expected lost entities and distance is measured in millimeters (Int3 coordinates [used in the simulation] have 1000 integers per floating point unit, and we interpret one unit as one meter), a rather larger value is needed for risk to play a significant role in pathfinding. We use a default value of 25,000 (2017, 240).

In order to maintain compatibility with the combat simulator, our implementation uses the same default value. This leads to some fairly high formation scores as a FOF will often have a large number of nodes. High formation scores are not necessarily an issue since formations are compared relative to each other. The large values for node threats do affect the weighting values for the other components of the scoring function. These weights often have to be significantly higher than the FOF weight, particularly as the size of the FOF increases. This ensures that the other components have an impact on the formation score and are not overridden by the FOF value.

2. Distance to Unit Leader

The penalty for being greater than the max distance away from the Unit Leader is calculated by subtracting the desired maximum distance value from the entity's actual distance from its unit leader.

3. Inter-Entity Spacing

The functions `calculateDispersion()` and `calculateCohesion()` in the Predictor Class handle the inter-entity spacing penalties. Pseudocode for each of these functions is shown in Algorithm 3 and Algorithm 4.

Algorithm 3. Calculating Dispersion Penalty

```
calculateDispersion()
    dispersion penalty = 0
    foreach (entity in the Unit)
        foreach (other entity in the Unit)
            distance = distance between entity and other entity
            if (distance > max desired distance)
                dispersion penalty += (distance - max desired distance);

    return dispersion penalty;
```

Algorithm 4. Calculating Cohesion Penalty

```
CalculateCohesion()
    cohesion penalty = 0
    foreach (entity in the Unit)
        foreach (other entity in the Unit)
            distance = distance between entity and other entity
            if (distance < min desired distance)
                cohesion penalty += (min desired distance - distance);

    return cohesion penalty;
```

4. Lines of Fire

If an entity is in a line of fire circular sector, they are assessed a set penalty value. This is designed to prevent entities from blocking each other's line of sight to targeting assignments and decrease fratricide. This penalty is applied for each line of fire the entity is in. The penalty is adjusted by the “In Line of Fire” penalty weight and is independent of the threat value on the node that the entity occupies.

A visual example of these factors affecting formation scoring is shown in Figure 22.

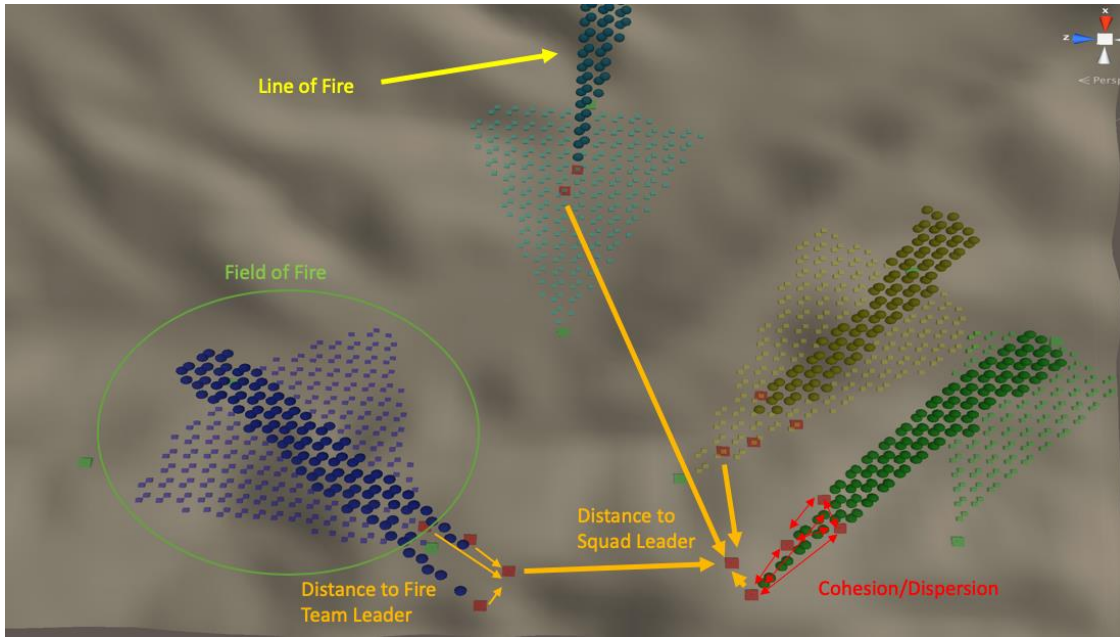


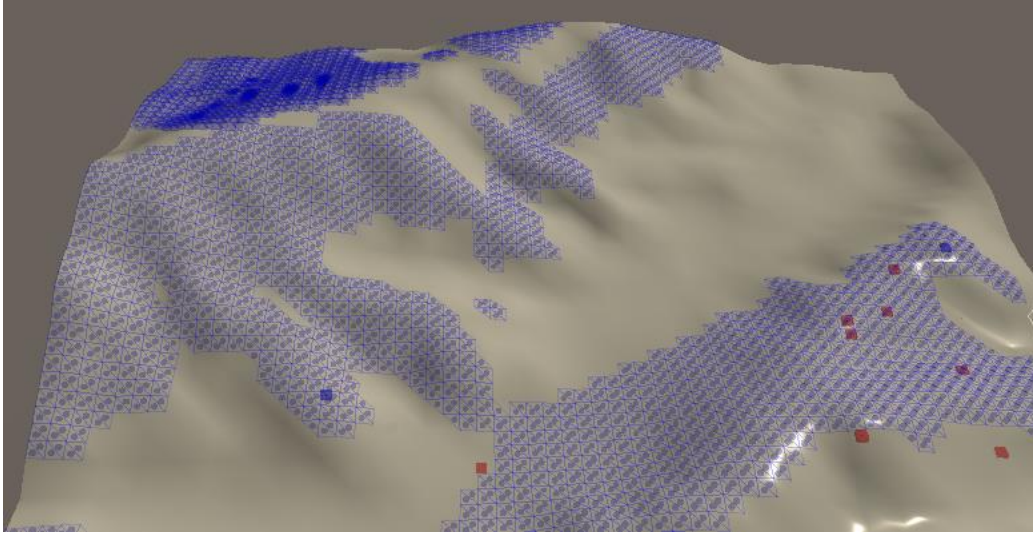
Figure 22. Formation Scoring Function Components.

This example shows a squad of three fire-teams and an observation post (OP). The squad is the top-level unit with each of the fire-teams and OP as subunits. The squad also has one entity, the squad leader. Each of the fire-teams consists of four entities and the OP has two entities. The fire-teams and OP all have one of their respective entities designated as fire-team leader. Each of the subunits has been assigned a FOF, shown by the light color-coded circular sectors. Entity lines of fire are shown by the darker nodes and are also color-coded similar to the assigned FOF. The formation score is a balancing act between all of the components. Fire-team entities try to maximize targetability into their assigned FOF, all while staying within range of their fire-team leader and maintaining good spacing with each other (cohesion/dispersion, shown in red). Fire-team leaders also try to maximize their FOF targetability, but they try to maintain range with the squad leader.

5. Cover

As stated previously, we use a second VisibilityGraph in the scene to map the node penalties with blue as the threatening force. This VisibilityGraph is built by placing representative blue entities on the terrain to capture red's estimation of where they will be attacked from. These entities are removed during simulation runs. Red's formation score

is penalized according to the threat value on the node by the representative blue forces. Figure 23 shows a VisibilityGraph for blue with two representative entities.



Note the actual Blue Force is behind the ridge in the upper-left corner.

Figure 23. Blue VisibilityGraph For Cover Calculations.

C. HILL CLIMBING

While the randomization of entity placement is handled at the Unit level, hill climbing is controlled by the Predictor class. The Hill climbing implementation is similar to Algorithm 2 in Chapter III. When evaluating entity position scores for neighboring nodes, only the threat for nodes for preferred targets and in the FOF are calculated. This reduces calls to the `rescan()` function, which calculates threat for every node in the VisibilityGraph. Detected entities will not randomize positions or hill climb, because their exact positions are already known. The Predictor class stores the original Force positions and formation score for comparison with the hill climbing results. Examples of hill climbing are shown in the following scenarios.

The first hill climbing example is shown in Figure 24 and Figure 25. In this scenario, a red squad of two four-entity fire teams and a squad leader are assigned two FOF to target. Fire-team 1 is assigned the green FOF on the right and fire-team 2 is assigned the teal FOF to the left. Line of fire and blue cover considerations are not included in the

formation scoring function for this scenario. The threat applied to each FOF by their assigned fire-team is shown by purple spheres. Larger spheres indicate a greater threat value.

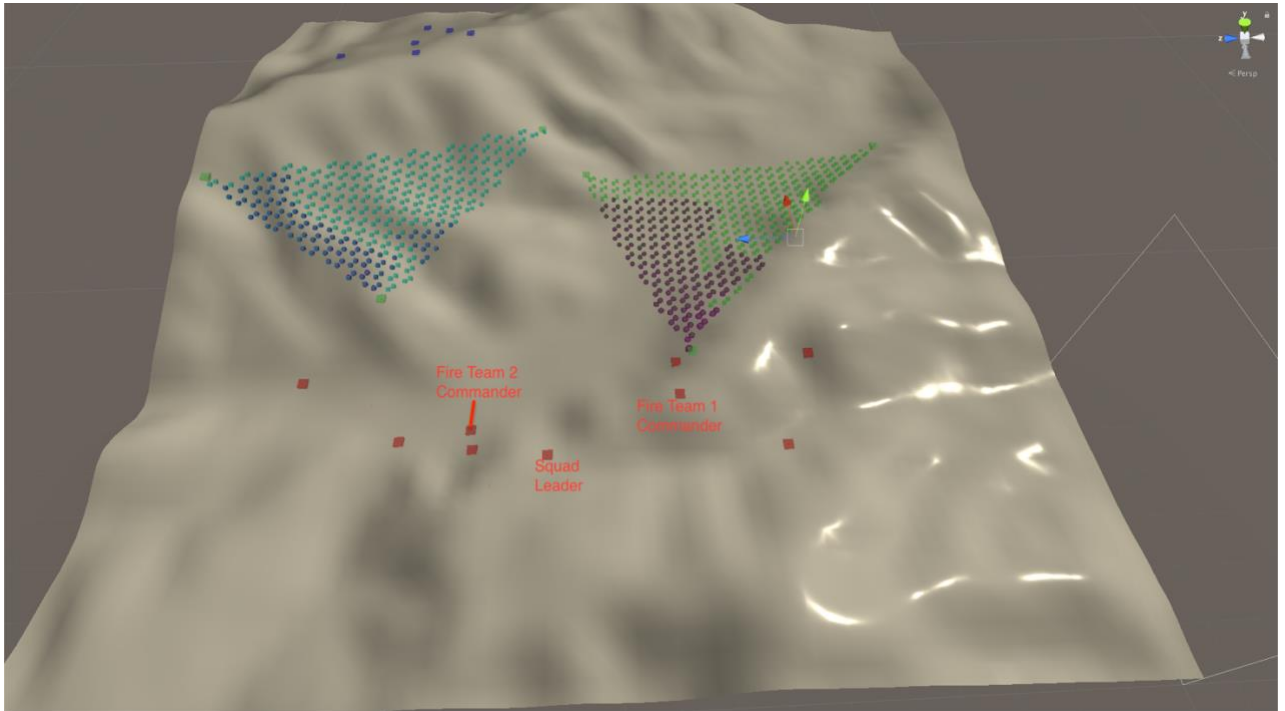


Figure 24. Simple Hill Climbing Example Starting Positions.

The original starting positions for the red squad are shown in Figure 24. Fire-team 1's FOF is a valley with fairly unobstructed views that are slightly threatened. Fire-team 2's FOF has a more undulating terrain that blocks the targeting of the right side. Spacing for both fire-teams is not bad, but there is a slight spread of individual entities.

The hill climbing results are presented in Figure 25. Threat values for almost all of the nodes in each FOF have increased, indicating improved targeting by the fire-teams. The squad leader has moved up to maintain spacing between the fire-team commanders. Fire-team 1 is almost able to target the entirety of its FOF, while pockets of fire-team 2's FOF remain untargetable due to terrain. The results do show an example of a slight imbalance on the weighting of components in the scoring function. The fire-team 2 commander and

one member of fire-team 1 are pulled slightly out of unit spacing and deeper into their assigned FOF. This indicates that the FOF weight may need to be reduced.

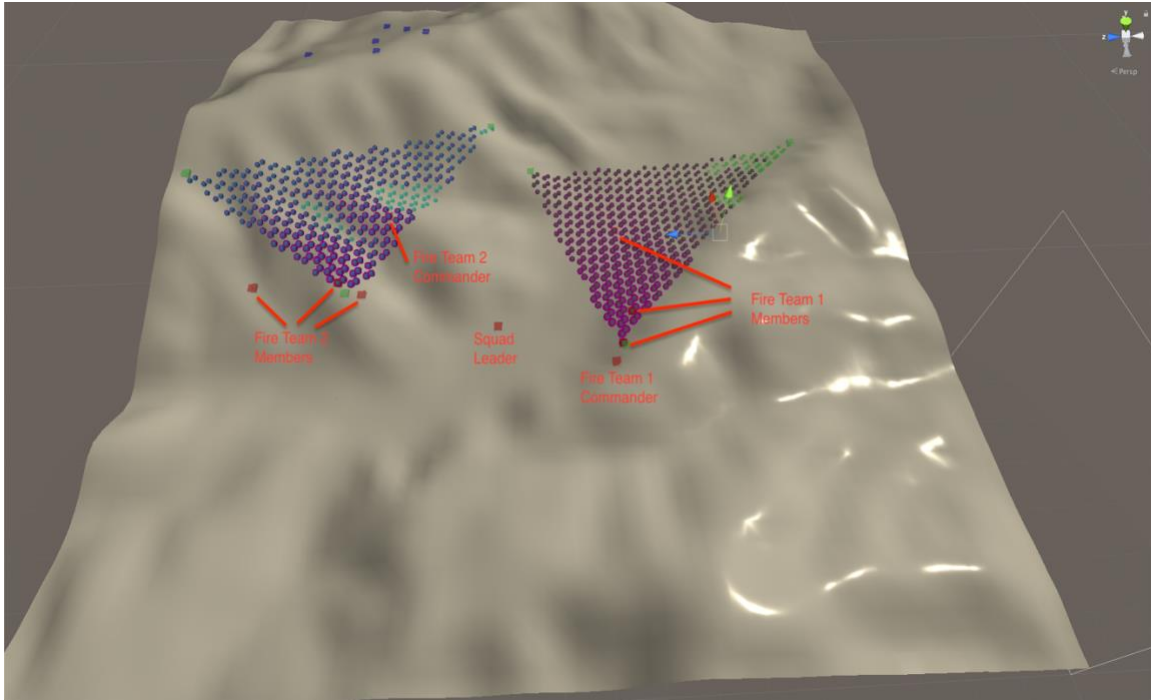
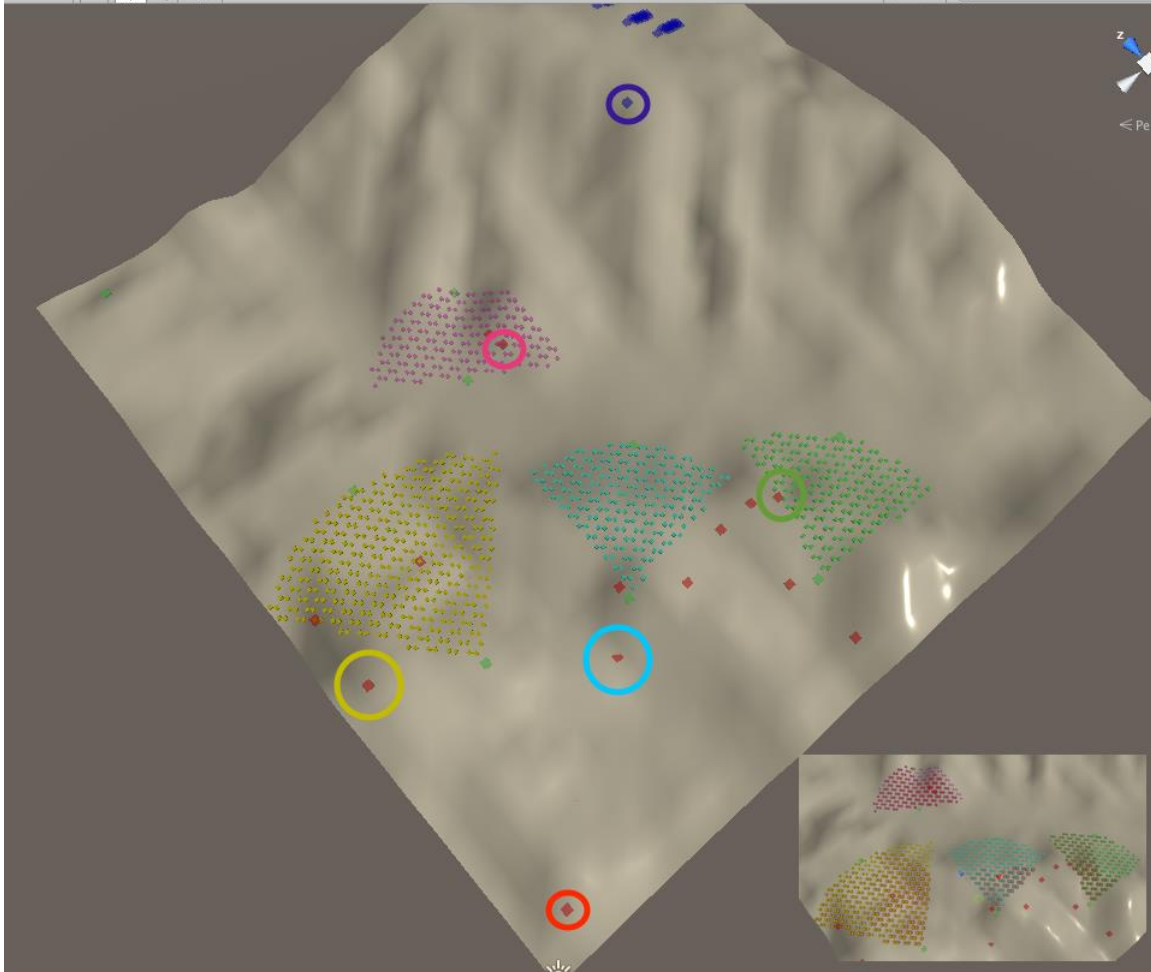


Figure 25. Simple Hill Climbing Example Ending Positions.

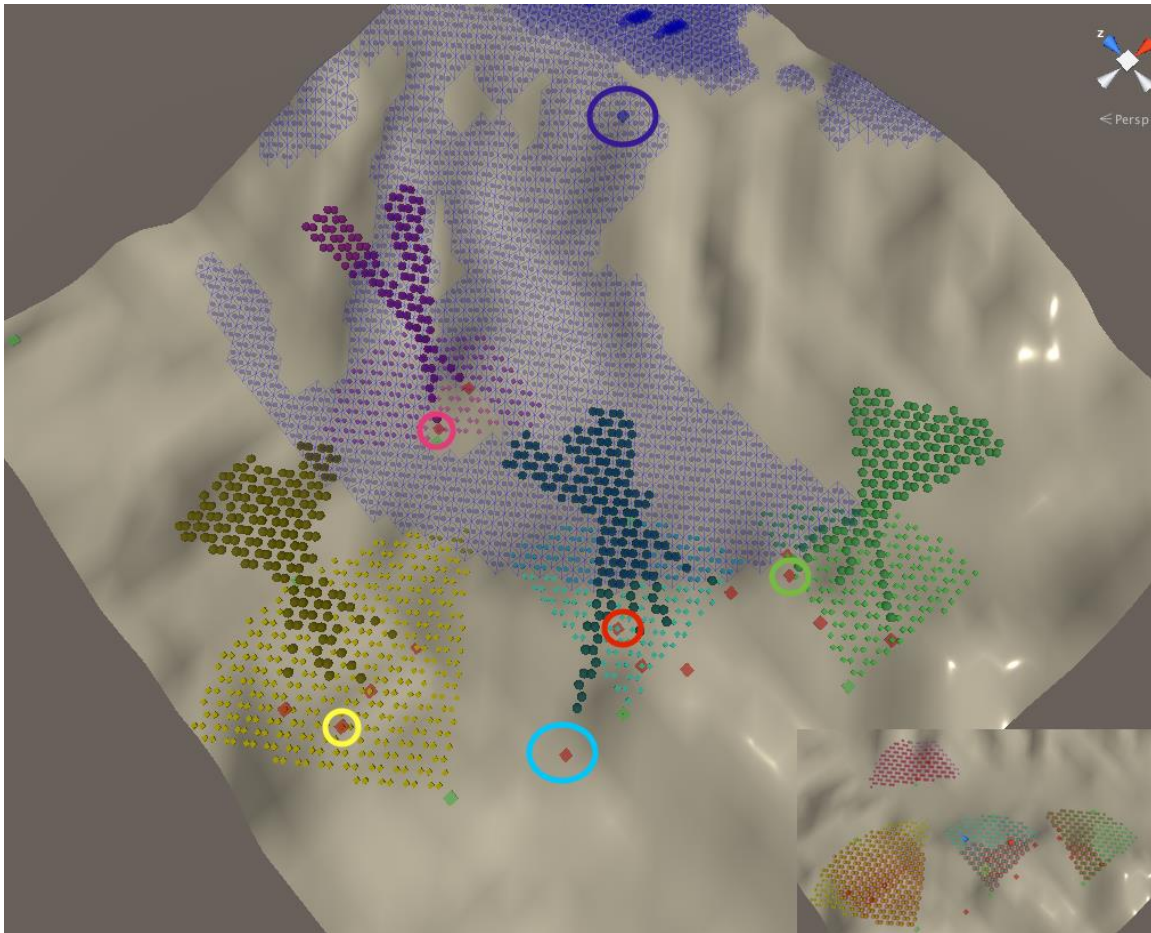
The second hill climbing example shows a more complex scenario with a scoring function that includes lines of fire and cover calculations. In this example unit spacing is exaggerated for better visualization. The scenario involves a red squad with three fire-teams and one OP. The starting random positions for the example are shown in Figure 26. The red squad leader is circled in red and the fire-team commanders are indicated by circles that reflect their assigned FOF. The blue forces are located behind a ridge at the top of the image, and a representative blue entity has been added (circled in blue) for calculating cover in the scoring function. Starting FOF targetability is also shown in the insert.



Fire-team leads are circled according to their FOF assignments. The red squad leader is at the bottom of the image. A representative blue entity for cover calculations is circled at the top. FOF targetability is shown in the bottom right insert.

Figure 26. Complex Hill Climbing Example Starting Positions.

The ending positions after hill climbing are shown in Figure 27. Squad geometry and FOF targetability have improved. The red squad leader has moved to a central location between the fire-team commanders. Entities have avoided areas that are not in cover. This is especially true of the purple OP unit and the green fire-team. Entities have also stayed out of each other's lines of fire. An added benefit to lines of fire visualization is it makes squad orientation easier to identify.



Lines of fire and blue threat on nodes have been added. Improved FOF targetability is shown in the bottom right insert.

Figure 27. Complex Hill Climbing Example Ending Positions.

One thing that may look off in Figure 27 is the position of the teal fire-team commander. He is further back than the other members of his fire-team. This illustrates how adding complexity to the force structure and scoring function can have some unintended effects. This entity is blocked from moving forward because it would put the red squad leader into its line of fire. Cohesion and line of fire penalties can often artificially block entities from moving to natural unit positions. This usually happens when areas of operation are overlapping causing entities from different units to be intermingled in the random placement step.

We use a rule-relaxing technique to help alleviate the entity blocking issue. In our hill climbing implementation we allow an initial round of climbing by the entities without cohesion or line of fire penalties. This helps entities move past any blockers. We then use a second round of climbing with the penalties included. It is important to note that rule relaxing does not completely remove local maxima and adding additional hill climbing rounds increases complexity.

D. GIBBS DISTRIBUTION

Now that we have demonstrated our methods for building a scoring function and using hill climbing to identify maximizing formations, we will show how the Predictor class implements the Gibbs distribution. The following sections demonstrate how the Predictor class builds its candidate formation set and then uses the Gibbs distribution to assign probabilities to each of the formations. These probabilities allow for the aggregation of risk from all formations in the candidate set into a single VisibilityGraph.

1. Building the Candidate Formation Set Distribution

Once targeting assignments and penalty weights have been set for the formation scoring function, building the candidate formations set distribution is completely handled in the Predictor Editor. Users indicate the desired number of formations in the candidate set by entering it into the Number Runs field. Buttons in the Editor are then used to perform the following steps:

a. Initialization

In this step precalculated visibility files are loaded for hill climbing and preferred targets and FOF are assigned to entities. Detected entities from the Held Unit array are marked to stay in current positions in the following steps.

b. Formation Generation and Score Calculation

The indicated number of formations are created and perform the randomization and hill climbing methods. When all the formations have completed hill climbing, users are able to cycle through each generated formation and see starting and ending positions, as

well as, their initial and final formation scores. Formation scores are stored in an array and the index of the highest scoring formation in the array is displayed in the Editor.

c. Calculate Probability for Formations

The probability for each of the generated formations is calculated according to the Formation Probability function in Chapter III. For the initial probability calculation, we found that using a T (temperature) value equal to the average formation score produced acceptable distributions for further refinement. The editor includes a button that adjusts T to the formation score average, or users can manually set the value in the Temperature field. The T value will be adjusted once the entropy calculations are completed.

d. Entropy Calculations

With the initial distribution created we can use the entropy functions to calculate and normalize the distribution entropy value ($\tilde{S}$). For the distribution calculated with T equal to the average formation score we generally found $\tilde{S} \approx 0.9$, with all formations in the candidate having close to the same probability.

In order to set T based on a desired value for $\tilde{S}$ the Predictor class uses the function `setTempertureBasedOnEntropy()`. This function makes small adjustments to the T value depending on if the current entropy is higher or lower than the desired $\tilde{S}$ value. After T is adjusted the distribution is recalculated and entropy is checked again. These adjustments continue until the distribution entropy is within a designated range of $\tilde{S}$. A range is used because of the difficulty in getting floating point values exactly equal. Pseudocode for the function is shown in Algorithm 3.

Algorithm 5. Setting temperature for a desired entropy

```
setTempertureBasedOnEntropy()
    entropyRange = a small value
    temperatureRange = a small value

    entropyHigh = entropy+ (entorpyRange * entropy);
    entropyLow = entropy - (entorpyRange * entropy);

    caluateDistribution();
    entropy = calculateEntropy();

    while(entropy < entropyLow || entropy >entropyHigh)
        if (entropy < entropyLow ){
            temperature = temperature + (temperatureRange* temperature);
        }else if(entropy >entropyHigh){
            temperature = temperature - (temperatureRange* temperature);

        caluateDistribution();
        entropy = calculateEntropy();
```

Figure 28 shows the Predictor Editor once all the steps for building the distribution have been completed. The example shows a five-formation candidate set. The green and yellow buttons (highlighted in the green box) allow the user to move through the different steps of building the distribution. Formation scores are stored in the Penalty Score array highlighted by the yellow box. Fields for temperature and entropy values are in the blue box. Formation probabilities are indicated by the red box.

With the distribution constructed, the Predictor Editor has a few extra options available to users. First, in addition to cycling through the formations in the candidate set, we are able to sample a single formation from the set according to the formation's probability. This is accomplished with the yellow button at the top of Figure 28. Second, we can save all of our distribution results and generated formations to a specified text file. As mentioned in our previous discussion on serialization, this is important because this information is not saved by simply saving the Unity scene, and it is also lost when we run the combat simulator.

The calculated distribution also gives us the ability to implement an uncertainty threat graph in WXXI. Uncertainty threat graphs are created by pushing the “Create Gibbs Graph” button in the Editor. The resulting VisibilityGraph is displayed on a sperate graph attached to the scene and node threat values are written out to a text file. This allows the uncertainty threat graph to be used as an alternate risk source for a VisibilityGraph in a combat simulation run.

Gibbs Sample Unit Position	
Cycle through Gibbs Runs	
Create Gibbs Graph	
Targeted Units	
Calculate Penalty	Read Visibility File
Visibility Check	Clear preferred Target Nodes
Output Gibbs results	Read Gibbs results
Output Penalty Scores	
Add Lines of fire	Clear Lines of Fire
Output Red Unit Positions	Read Red Unit Positions
Initialize	
1. Generate Random Formations and Calculate Scores	
2. Set Temperature to Avg Score	
3. Calulate Distribution	
4. Calulate Entropy	
Set Temp Based on Entropy	
Target Unit	EnemySquad (Unit)
Opposing Unit	InfantryPlatoon (Unit)
Gibbs Index	0
Alternate Graph File Name	entropyGraph
Gibbs Results File Name	EntropyResults
Blue Starting Position File Name	StartingBlueUnitPositions
Blue Position File Name	BlueCoverCalculationPositions
Random Spawn Range	X 200 Y 0 Z 200
Random Spawn Off Set	X 0 Y 0 Z 0
Astar Graph Index	1
Enemy Graph Index	2
Gibbs Graph Index	3
Calculated Penalty	3.439218e+10
Number Runs	5
Penalty Scores	
Size	5
Element 0	1.178735e+10
Element 1	-5.892074e+09
Element 2	1.838941e+10
Element 3	1.706132e+10
Element 4	-7.728154e+09
Starting Penalty Scores	
File Read	☒
Temperature	4.437021e+11
Entropy	0.9998003
Highest Penalty Index	2
Expo Penalty Divided By Sum	
Size	5
Element 0	0.2022304
Element 1	0.1943309
Element 2	0.2052619
Element 3	0.2046485
Element 4	0.1935284

Figure 28. Results From Building the Distribution.

In the next section, we will show how the results from the above example are used to construct an uncertainty threat graph.

2. The Uncertainty Threat Graph

The VisibilityGraph for each formation from the above example are shown in Figure 29. Formations are named according to their index position (0-4). As mentioned previously, the red triangles represent nodes visible to the red forces and the spheres represent the node's threat value. The sphere's size increases as threat increases.

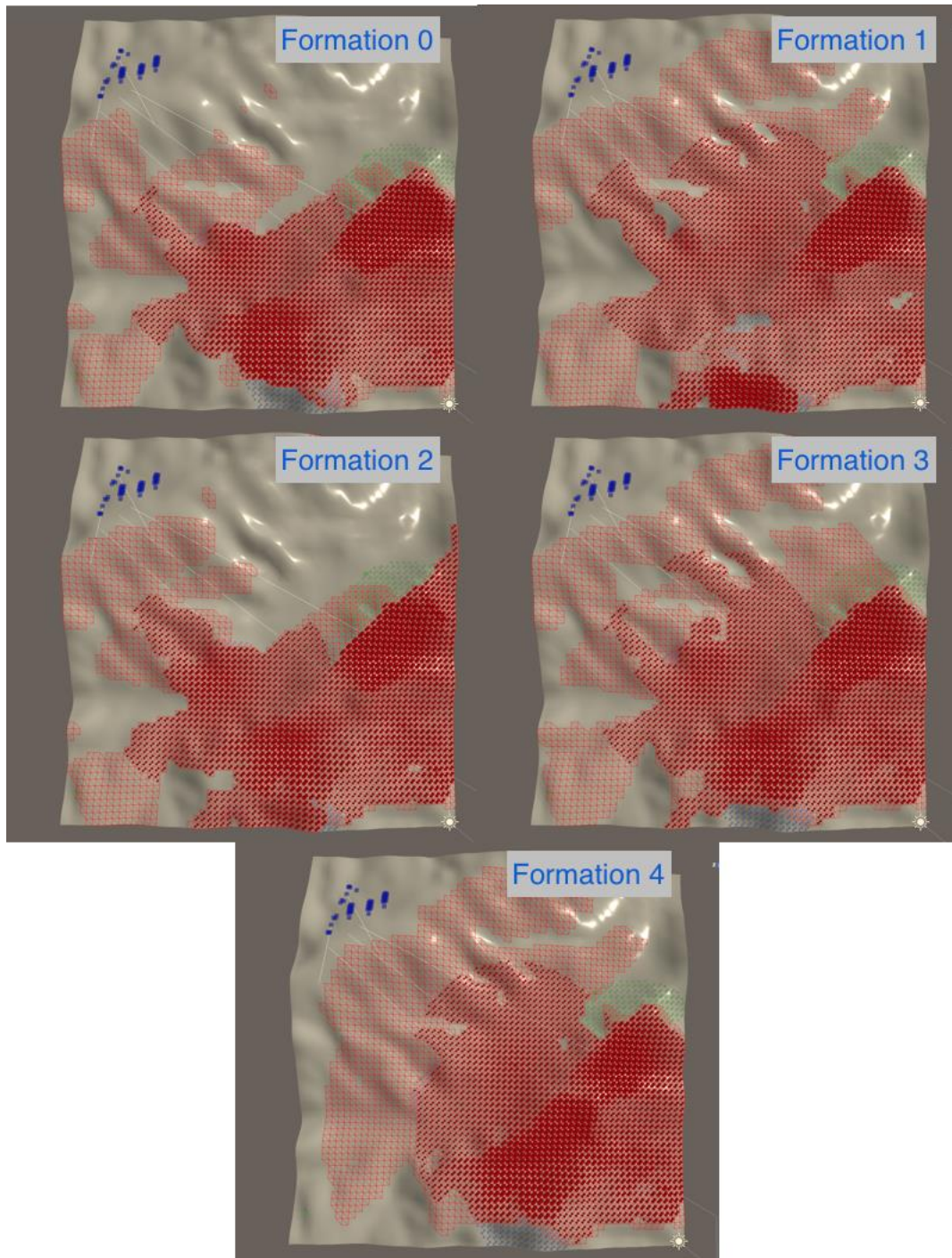
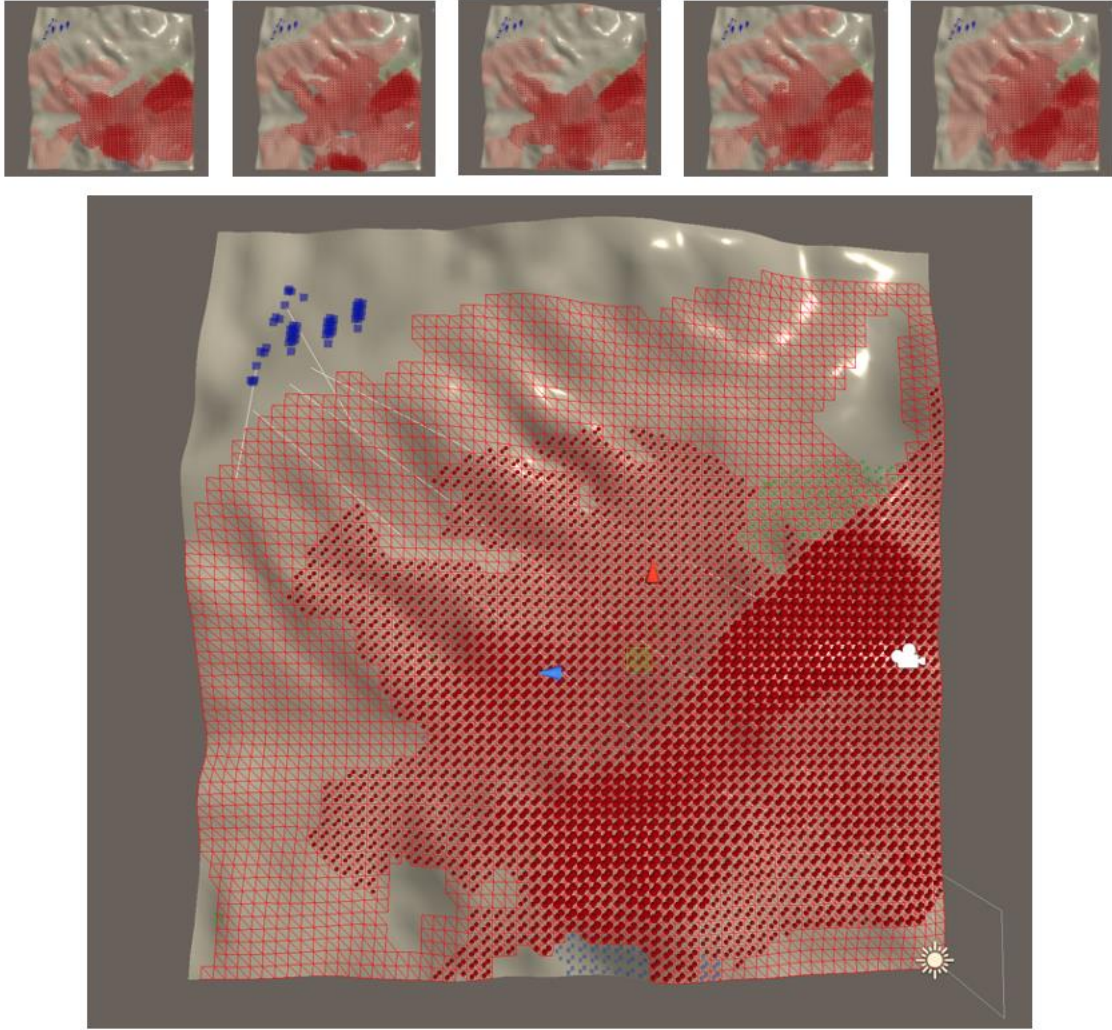


Figure 29. VisibilityGraphs for Formations in the Distribution.

The formation scores for each of the formations in Figure 29 are reflected in the yellow box in Figure 28. Each formation consists of a red squad comprised of three four-man fireteams defending the ridge, and a two-man OP in the middle of the map. It is important to note that

Formations 1 and 4 both received negative formation scores while Formations 2 and 3 were the highest scoring formations. The low scores for Formations 1 and 4 were due to poor placement of the OP members and too many entities not finding cover.

Looking again at the Editor in Figure 28, we can see that entropy (in the blue box), is set at 0.9998. Since entropy is almost equal to one, the formation probabilities (in the red box) are almost equal to each other at about .20 for each formation. Given that all formations have an almost equal probability, the resulting uncertainty threat graph reflects elements found in each of the individual Visibility Graphs. The resulting uncertainty graph is shown in Figure 30.



The original VisibilityGraphs are above the resulting uncertainty threat graph.

Figure 30. Uncertainty Threat Graph for Entropy = 0.9999.

3. Entropy Effects

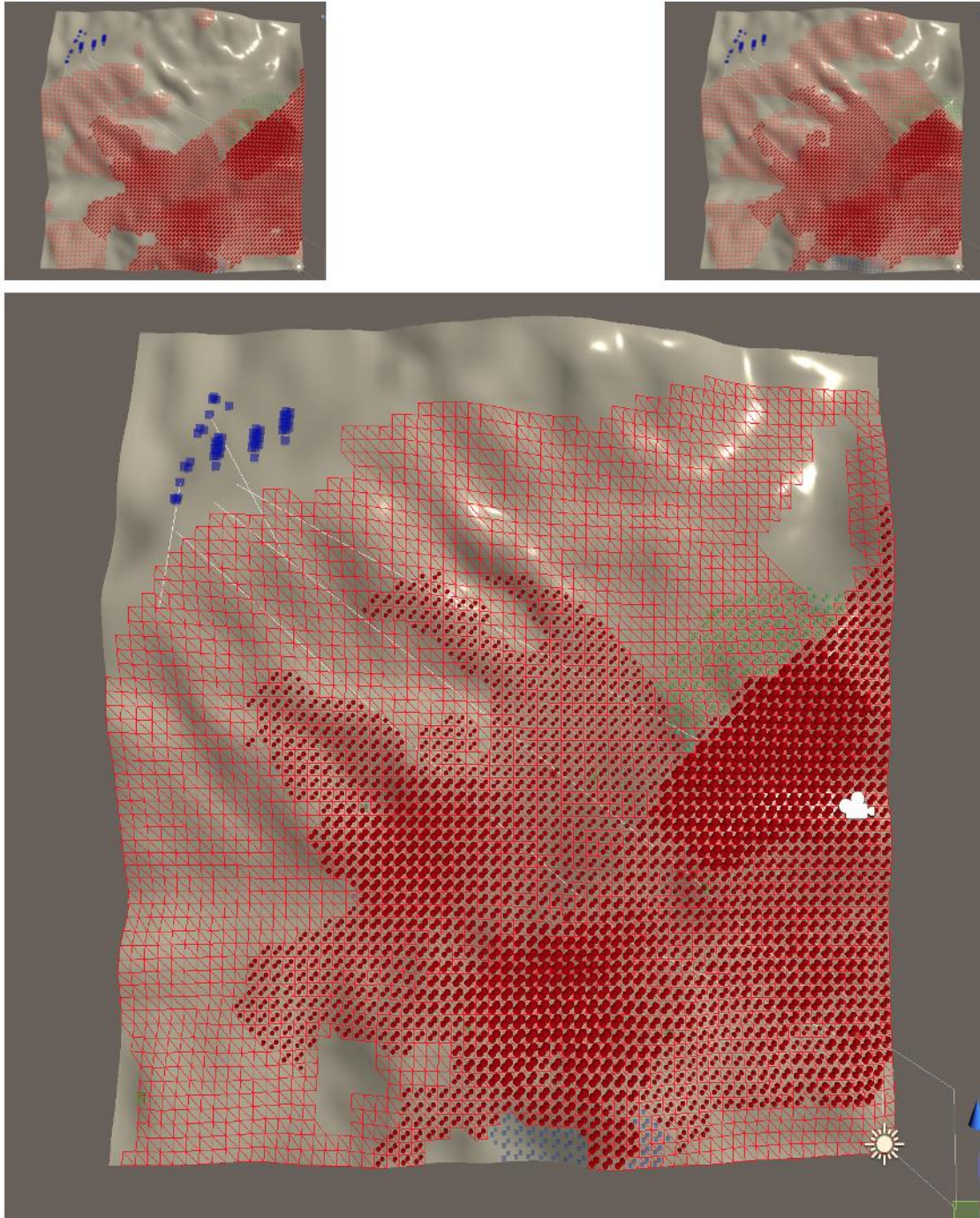
We can visually demonstrate the effect entropy has on the distribution and uncertainty threat graph by decreasing its value in the above example. Results for decreasing entropy to 0.5 are shown in Figure 31.

Penalty Scores	
Size	5
Element 0	1.178735e+10
Element 1	-5.892074e+09
Element 2	1.838941e+10
Element 3	1.706132e+10
Element 4	-7.728154e+09
Starting Penalty Scores	
File Read	☒
Temperature	2.469765e+09
Entropy	0.5000463
Highest Penalty Index	2
Expo Penalty Divided By Temp	
Expo Penalty Divided By Sum	
Size	5
Element 0	0.04175844
Element 1	3.250246e-05
Element 2	0.6048946
Element 3	0.3532991
Element 4	1.545438e-05

Formation scores are in the yellow box. Entropy and temperature are in blue and formation probabilities are in red.

Figure 31. Distribution Results for Entropy = 0.5.

The drop in entropy makes the probabilities more dependent on formation scores. In the candidate set, Formation 2 had the highest formation score, followed by Formations 3 then 0. With entropy equal to 0.5, probability for Formation 2 is 0.60 and for Formation 3 it is 0.35. The probabilities for Formations 1 and 4 are both almost 0. Since the bulk of the candidate set probability is allocated to Formations 2 and 3, the resulting uncertainty threat graph mostly reflects the features found in their respective VisibilityGraphs. This is shown in Figure 32.



VisibilityGraphs for Formation 2(left) and Formation 3(right) are shown above the resulting uncertainty threat graph.

Figure 32. Uncertainty Threat Graph for Entropy = 0.5.

E. PLAN RECOGNITION

We developed the PlanPredictor class to handle our implementation of the plan recognition methods. This class is separate from the Predictor class and has its own editor. The PlanPredictor class enables targeting points and formation scoring values to be written to and read from text files. This allows different formation scoring functions to be swapped in and out of scenes faster and reduces the need to rebuild functions from scratch. This helps to make our scoring functions behave like true adversary templates that can be saved and loaded on to different terrains as needed. The PlanPredictor does still rely on the Predictor class to make the targeting assignments and designate detected entities in the initialization step once a new plan has been loaded. The PlanPredictor Editor is shown in Figure 33.

Similar to the Predictor Editor, the PlanPredictor Editor includes buttons for calculating and setting entropy and temperature. However, the plan probabilities in the PlanPredictor Editor are handled differently than the formation probabilities in Predictor Editor. In the Predictor, probability calculations are done on a complete candidate set in one step. For the PlanPredictor, new formation scoring functions are loaded, initialized, and then evaluated for known entity positions. The results are then added to an existing array of scores and the distribution recalculated.

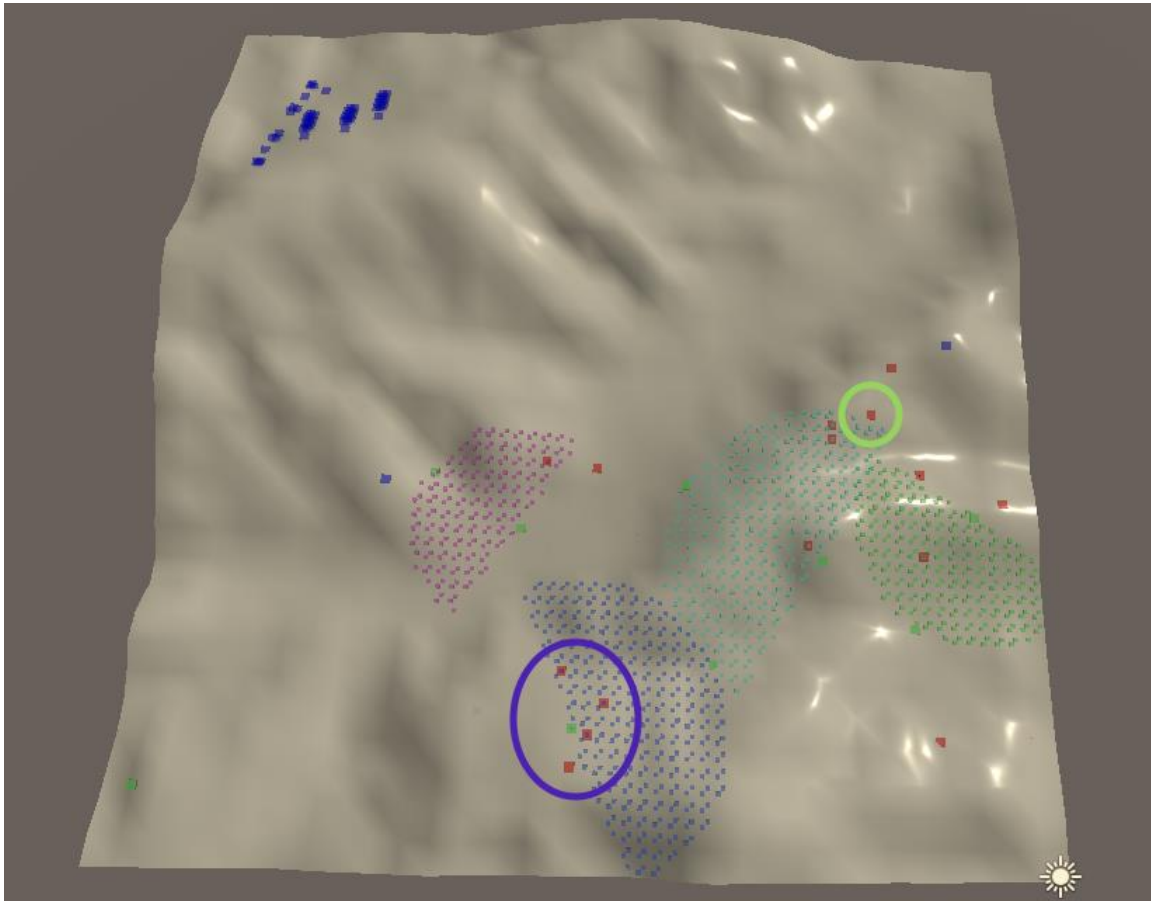
OutputPlan		InputPlan	
Calculate Penalty for Held Members			
Update Plan Distribution			
Set Temperature to Avg Score			
Calculate Distribution			
Calculate Entropy			
Set Temp Based on Entropy			
Script	PlanPredictor		
Predictor	Predictor (Predictor2)		
Target Unit	None (Unit)		
Fof Points	Field of Fire Points (Unit)		
Input Plan File	ridge		
Output Plan File	output		
Temperature	7.680353e+09		
Entropy	0.740131		
Penalty Scorefor Held Members	-1.713562e+10		
▼ Penalty Scores			
Size	3		
Element 0	3.895936e+09		
Element 1	-1.27022e+10		
Element 2	7.527395e+09		
▼ Expo Penalty Divided By Temp			
Size	3		
Element 0	1.660735		
Element 1	0.1913106		
Element 2	2.664681		
▼ Expo Penalty Divided By Sum			
Size	3		
Element 0	0.3676854		
Element 1	0.04235604		
Element 2	0.5899585		
Number Of Plans Evaluated	3		

Plan probabilities are highlighted by the red box.

Figure 33. The Plan Predictor Editor

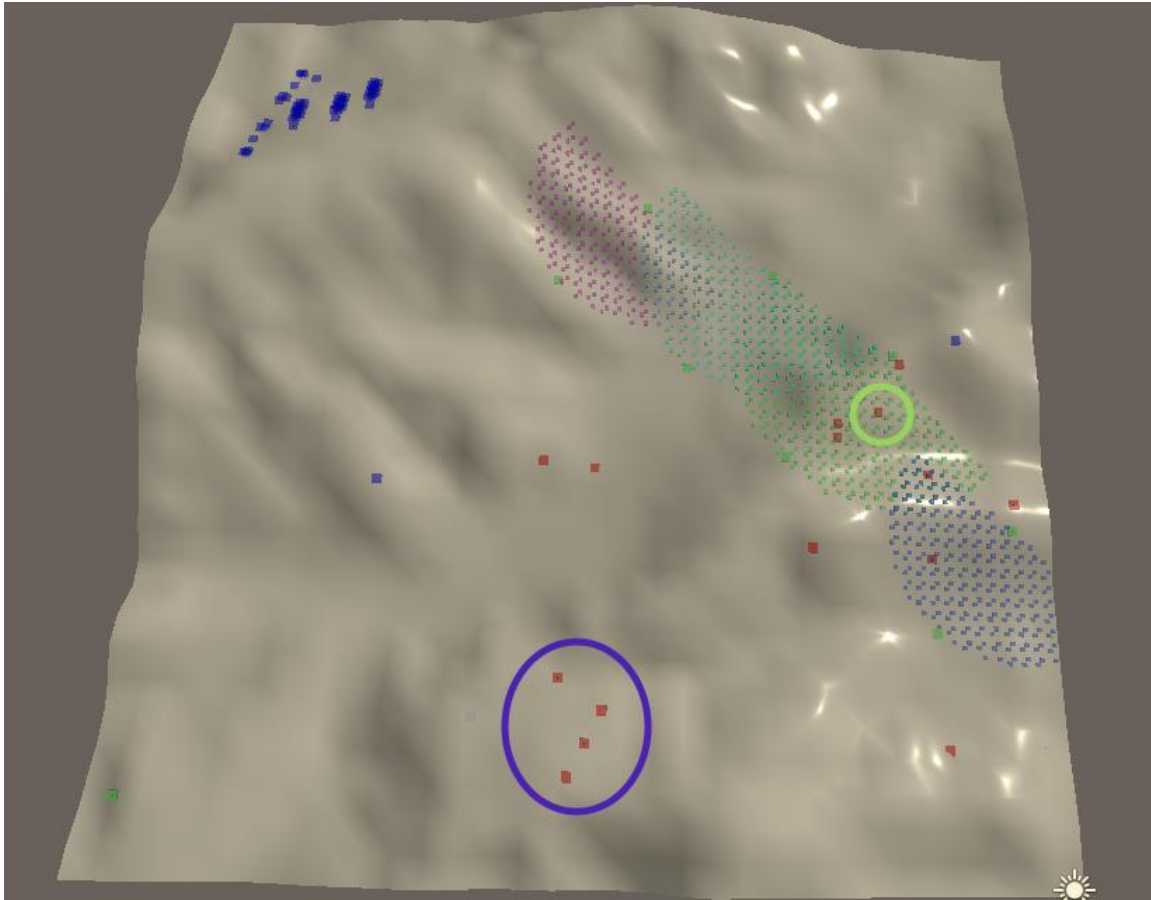
The results highlighted by the red box in Figure 33 come from the evaluation of three possible formation scoring functions (plans) against five detected red entities. The entities include a fire-team commander and a separate full fire-team. The detected entities are circled in each image according to their assigned FOF. The three plans include a box

defense around the squad leader (Figure 34), an ambush (Figure 35), and a ridge defense (Figure 36). Of three possible scoring functions, ridge defense (Element 2 in Figure 33) and box defense (Element 0) have the highest probability of being the correct plan. The particular qualities of these plans will be discussed as part of our evaluations in the following chapter.



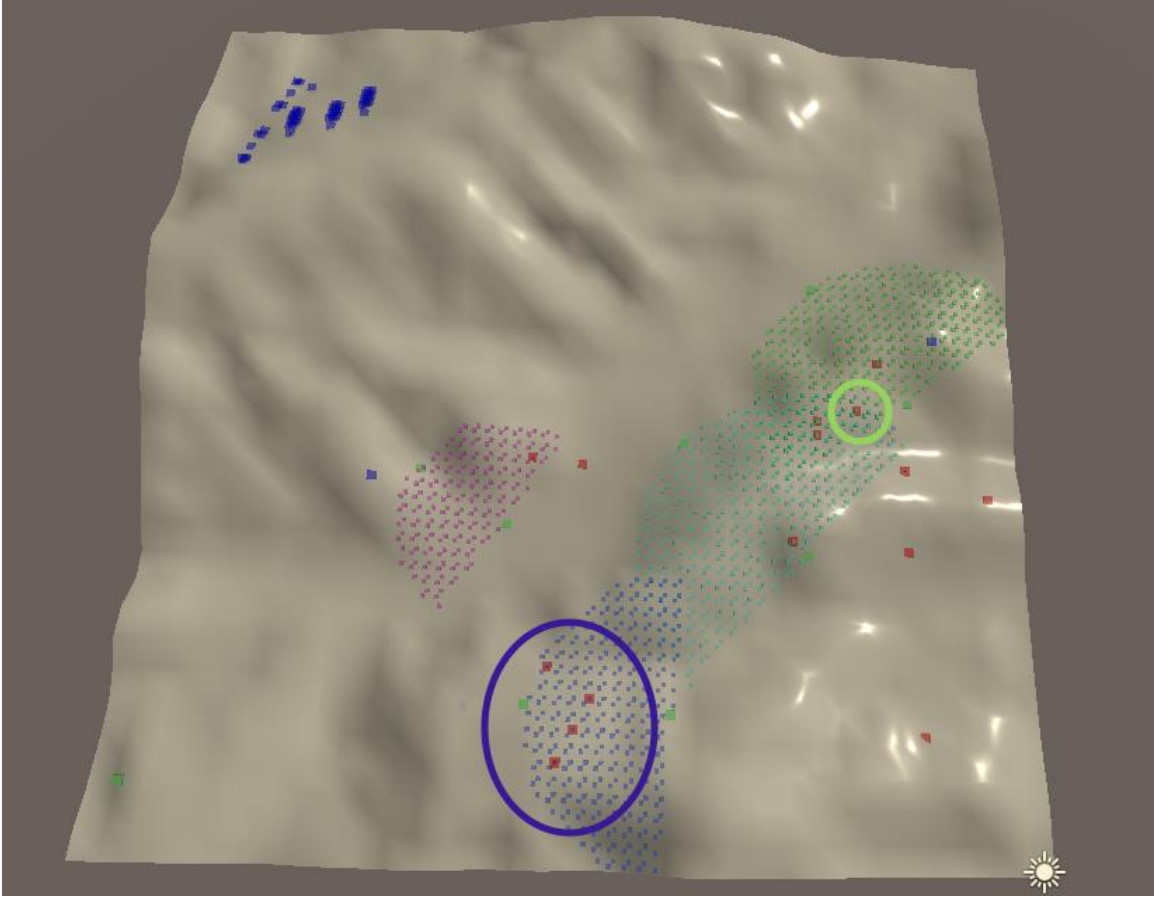
The probability for this plan given the known positions of the circled entities is 0.37 (See Figure 33, Element 0 in the highlighted probabilities).

Figure 34. Box Formation Scoring Function Assignments.



The probability for this plan given the known positions of the circled entities is 0.04 (See Figure 33, Element 1 in the highlighted probabilities).

Figure 35. Ambush Formation Scoring Function Assignments.



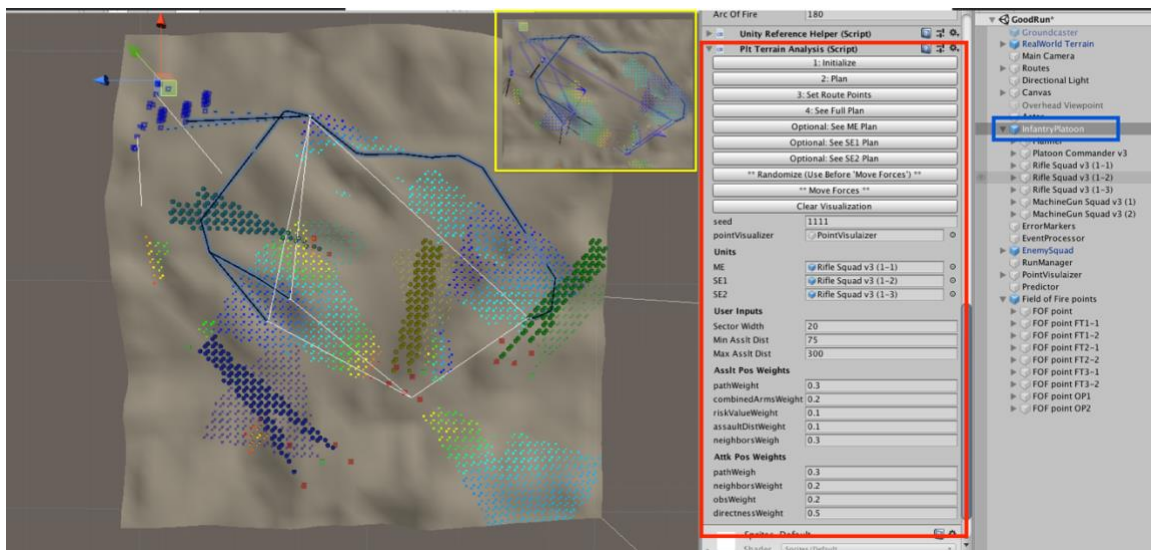
The probability for this plan given the known positions of the circled entities is 0.59. (See Figure 33, Element 2 in the highlighted probabilities.).

Figure 36. Ridge Defense Formation Scoring Function Assignments.

F. INTEGRATION WITH OTHER MODULES

As previously discussed, one of the great advantages of using a Unity-based CSE is the ability to integrate other modules for increased functionality. In developing our prediction method implementations, we have limited our alteration of existing code as much as possible to ensure the proper function of Harder's CSE and fire support planner (2017). We were able to integrate Severson's terrain reasoning and maneuver planner package that was also developed for WXXI (2019). Severson's planner is a script that attaches to the attacking (blue) force in a scene. It allows for automated creation of routes for the blue force maneuver element. Like Harder's fire support planner, it also needs exact opposing force positions.

While we did not attempt to “blind” Severson’s planner as we did to Harder’s, we were able to test its functionality against formations generated by the Predictor. We were then able to execute the plans generated by Severson’s terrain reasoning module in the combat simulator. Figure 37 shows an example of Severson’s terrain reasoning package planning against a Predictor formation. In the example, Severson’s planner automatically sets the objective points for the attacking blue force to the center point of red’s defensive position. The resulting fire support plan generated by Harder’s planner is also displayed in the yellow insert.



The Terrain Reasoning Editor (red box) is attached to the Infantry Platoon GameObject (blue box). Red force lines of fire are color-coded to their assigned FOF. Output from the fire support planner is shown in the insert (yellow box).

Figure 37. The Terrain Reasoning Module.

G. SUMMARY

In this chapter, we demonstrated the implementation of our prediction methods in the WXXI CSE. We also showed how we were able to integrate our techniques with the existing WXXI functionality. In the next chapter, we will use the WXXI combat simulator to perform a quantitative evaluation of the Predictor.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EVALUATION

This chapter presents our evaluations of the implementations presented in Chapter IV. We begin by describing how components of the Predictor class were tested for proper functionality. Then, we define the base battle set-up used in our evaluation scenarios, as well as its applicable indicators to measure performance. The chapter then presents three evaluation scenarios. In the first scenario, we evaluate how the Formation Score is related to battle outcomes. Next, we provide a demonstration of battle planning with different levels of intelligence about enemy locations. Finally, we show blue planning under fog-of-war conditions.

In developing his fire support planner, Harder focused on three components of testing: functional, quantitative, and qualitative (2017). We utilized these same focus areas for our evaluations. Functional testing ensures that each piece of the program is working as intended and produces accurate results. Once we have determined components are functioning properly, “we test for positive or negative changes in the quantitative results” (Harder 2017, 273). For our testing, quantitative measurements usually entailed the statistical results from multiple battles fought by blue and red forces in the simulator. In addition to these numerical results, we looked at the qualitative behavior displayed by both the blue and red forces. This includes red’s ability to construct reasonable formations with a scoring function and blue’s ability to plan when given different levels of information.

A. COMPONENT TESTING

In our development of the Predictor and PlanPredictor classes, we found the visualization tools available in WXXI invaluable for component testing. Harder modified the Astar Pathfinding Project visual representations of nodes for his own development and testing of the VisibilityGraph (2017). We made further modifications to allow for distinguishing a unit’s FOF, lines of fire, and threat into targeting assignments. In addition to providing a graphic representation of the formation scoring function components, these modifications allowed quick visual conformation that units were hill climbing correctly. We were also able to check our precalculated visibility table by comparing graphs created

by the VisibilityGraphPre class with those created using the original VisibilityGraph class. Instances of all of these visualizations can be seen in many of the implementation examples in Chapter IV.

Checking the numerical accuracy of functions was usually accomplished with outputs to the Unity console window or fields in the Editor. This was useful in the initial testing of the hill climbing method and was employed in an experiment for our previous paper. We performed 1000 rounds of random placement and hill climbing using the simplified scoring function from the scenario shown in Figures 24 and 25 (Maroon and Darken, 2019). In addition to seeing visual improvements in most of the formations, we found that average formations scores rose significantly and the standard deviation of scores decreased. Table 1 provides full results from all of the runs.

Probability and entropy calculations were checked both numerically and by visually inspecting the uncertainty threat graph for expected features.

Table 1. Random Placement and Hill Climbing Test Results. Source: Maroon and Darken (2019).

	Starting Score	Ending Score	Change %
μ	12.320	38.414	251
σ	4.483	7.757	146
σ/μ	0.364	0.202	-55

Formation Scores are scaled by 1/109.

Once we were confident that the components of the Predictor class were functioning correctly, we looked to perform further evaluations using WXXI's combat simulator.

B. BATTLE SCENARIO

In building our evaluation battle scenario we wanted to create an engagement that would be able to address the following questions.

- How does the formation score of red's forces affect the outcome of the battle? Is it reasonable to expect that higher formation scores indicate a more favorable outcome for red's defense?
- How do blue's tactics and battle outcomes compare when planning with different levels of knowledge about red positions? What is the difference between using observed risk calculations, estimated risk calculations, and perfect information?
- Can we replicate a battle fought with fog-of-war effects in WXXI? What does blue's strategy look like when it is only aware of detected red entities?

Since the battle tactics for blue's forces would be controlled by Harder's fire-support planner, we based our evaluation scenario on a modified version of his blue platoon vs red squad battle on the Cayucos Creek terrain (2017). This had several advantages. First, it freed us from having to come up with a behavior set for the attacking forces. Second, it allowed us to use prefabricated unit hierarchies and weapons templates. Finally, we were familiar with the features of the Cayucos terrain, which allowed us to use our previous visibility calculations and gave us a better understanding of the adjustments that were needed to suit our evaluation goals.

1. Scenario Modifications

There were some limitations to the original scenario that we had to modify or account for. First, in the original scenario, blue was tasked with a deliberate attack to clear a known red position. If we were to keep this tasking for our testing, we would have to change blue's tactical control points for each new red formation, adding extra variability between battles. Since we wanted to maintain consistency with blue's tasking, we modified their objective to only kill the red squad leader located at the corner opposite their starting position. This allowed us to keep blue's assigned tactical control points consistent and, since only the red squad leader had to stay in a specific place, gave us greater freedom in how we arranged the red defense.

The second issue we encountered was that when controlled by the fire-support planner, the blue force was very good at avoiding the red formation completely. To combat this, we allowed greater spacing of red entities in the formation score, so that red's defense covered more of the map. Additionally, for some tests we positioned blue's tactical control points to specifically encounter all elements of the red formation.

2. Scenario Force Structure and Armament

We did not modify Harder's original force paring of a reinforced blue platoon assaulting a dug-in red rifle squad. The red rifle squad has 15 total entities arranged in three four-man fire-teams, one two-man OP, and a squad leader. All entities in the rifle squad are armed with rifles. The blue platoon has a platoon commander, three thirteen-man rifle squads (no OP units for these squads), and two seven-man machine gun squads for a total of 54 entities. All members of the platoon have rifles, except two members of each machine gun squad have a machine gun. The machine guns have a range of 1800 meters, the rifles only have a range of 550 meters (Harder, 2017, 286). Since the machine gunners can target almost anywhere on the terrain that is visible to them, they provide fire support to the assaulting blue rifle squads. While blue significantly outnumbered red in the scenario, Harder notes that this is an expected numerical advantage for an attacking force in military doctrine (2017).

3. Simulator Output and Data Collection

We were able to collect both qualitative and quantitative data from our combat simulator runs. The combat simulator provides visualization of blue force control measures, pathfinding, fire support tasking, and suppression effects. An example of a battle in progress is shown in Figure 38.

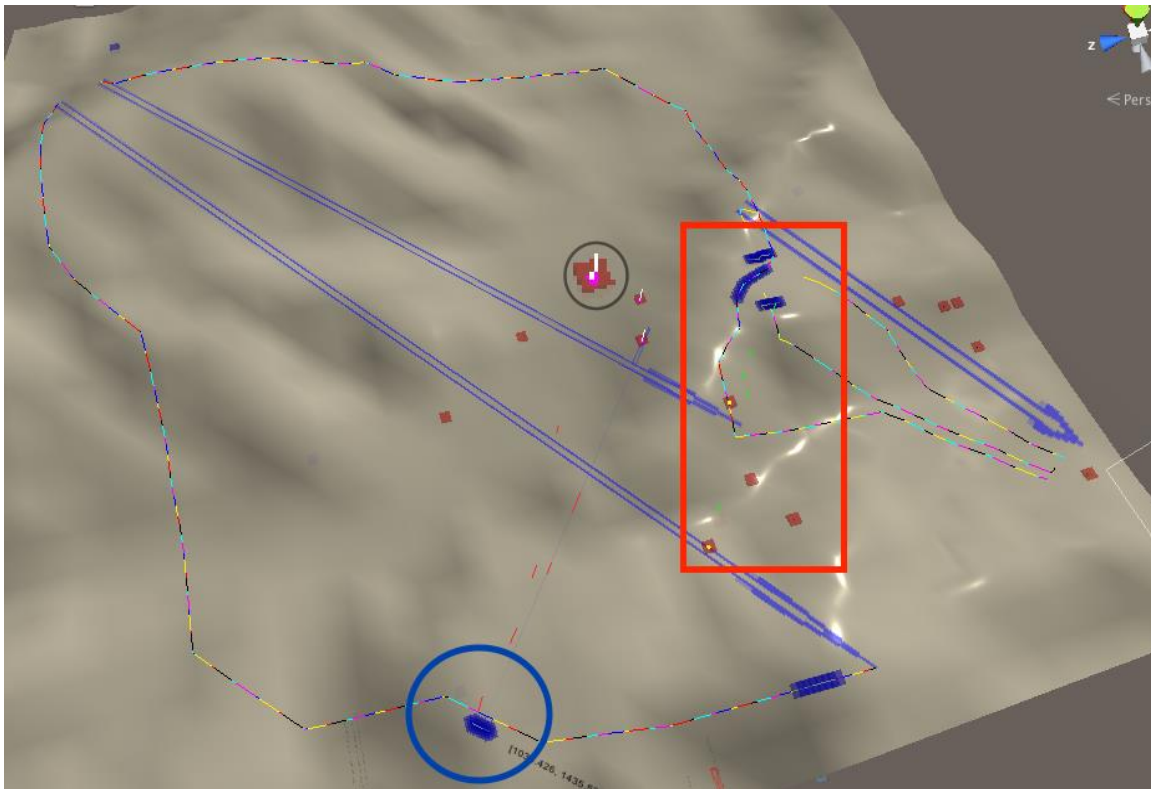


Figure 38. WXXI Combat Example.

In this example, the straight blue double arrows indicate where a blue unit is moving to, while the curving multicolor line shows the actual path that the unit takes. The blue circle outlines a machine gun squad suppressing a red fire-team. The squads' fire support tasking is indicated by the single thin blue and their shots shown by the red line segments. The suppression effects on the red fire-team are shown as white lines above the red entities (magnified in the black circle). In the red box, a fire-team is attacking the advancing blue assault force. Their shots are shown as green line segments.

These visualizations helped us understand how blue would respond to different formations and were useful in developing the battle scenario and red scoring functions. Additionally, we were able to overlay the VisibilityGraphs being used by the simulator, so we were able to see how blue's view of risk changed its tactics. Figure 39 shows the same moment in battle as Figure 38, but with the VisibilityGraph added. With blue's risk map visible, it is easier to understand the pathfinding decisions of its units.

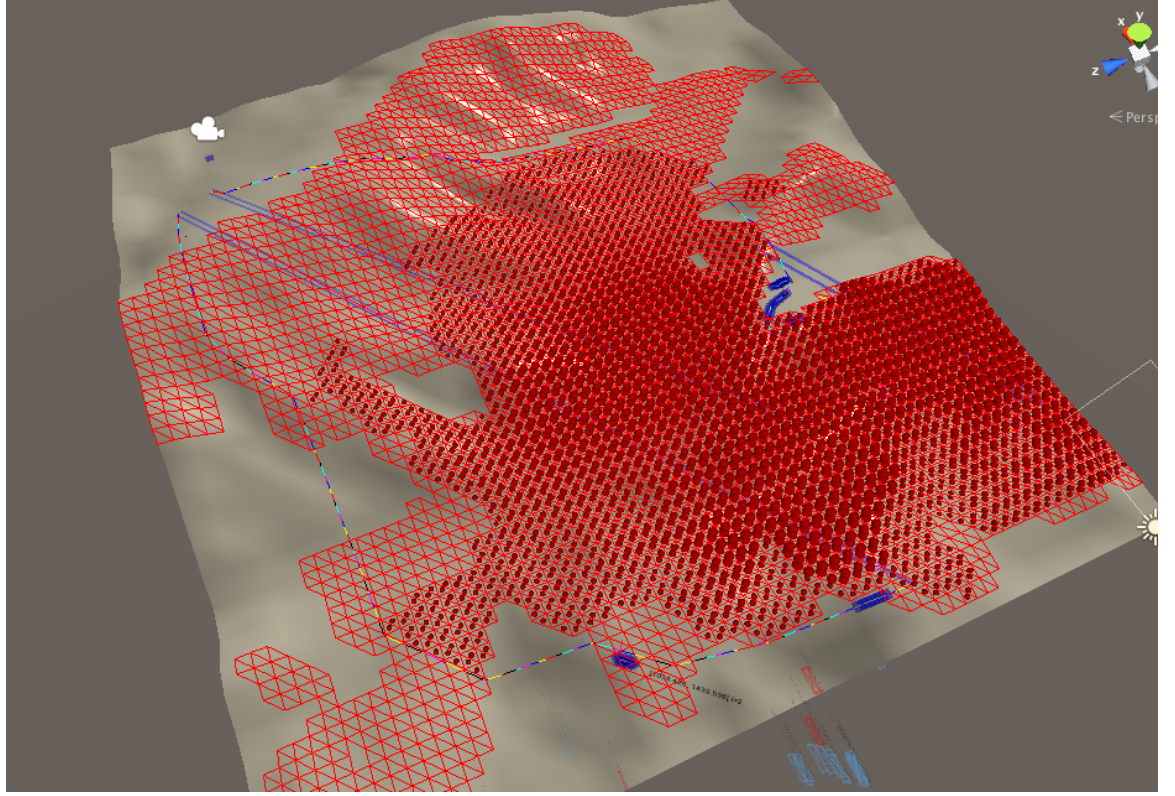


Figure 39. WXXI Combat Example With VisibilityGraph

We collected numerical data using the `MultipleRunManager` class discussed in Chapter II. For our evaluations, we looked at the end strength for blue and the survival rate for red forces. We also used Harder’s (2017) implementation of Helmbold and Khan’s fractional exchange ratio (FER) given as:

$$FER = \frac{\frac{Red(t_{start}) - Red(t_{END})}{Red(t_{start})}}{\frac{Blue(t_{start}) - Blue(t_{END})}{Blue(t_{start})}}$$

In the FER equation, $Red(t_{start})$ and $Blue(t_{start})$ are the number of entities in each respective force at the start time (t) of the simulation run. $Red(t_{end})$ and $Blue(t_{end})$ are the remaining entities for each force at the end of the run (Harder 2017).

Harder also defines a mission accomplishment score that is a function of the number of objectives completed by the blue force divided by the number of objectives assigned

(2017). Since our scenario only has one objective (kill the red squad leader) this value is either 1 or 0.

When evaluating red's performance, we looked to see how many blue entities were killed and if their defense prevented the squad leader from being assassinated. We also felt that it would also be reasonable that red would want to minimize the FER, since that would indicate a more positive outcome for them.

C. FORMATION SCORING EVALUATION

For our initial evaluation, we used the battle scenario to determine if higher formation scores for red's defensive laydown of forces would lead to improved outcomes for red. We conducted this experiment using a candidate set of 50 formations created using our hill climbing algorithm. Each formation was saved to an individual Unity scene and fought 25 battles using the MultipleRunManager. For these battles, Harder's fire support planner was allowed to run without any impediments, giving it full knowledge of red positions. While blue's maneuver and fire support actions would change for each of the 50 formations created, they would be the same for the 25 battles fought against a particular formation.

1. Scenario and Formation Score Specifics

We tailored the base battle scenario to ensure a thorough engagement of all red's defense by blue. Figure 40 shows the blue starting positions, routing, and objective. The attacking platoon starts behind a ridge, indicated by the blue circle in the top left of Figure 40. Blue's maneuver elements (three rifle squads) move across the valley to the waypoints indicated by the white circles. These waypoints were positioned to limit blue's ability to by-pass the red defense and guarantee that each of red's fire-teams had some contact with the attackers. The objective of blue's attack, the red squad leader, is placed at the bottom right of the map and indicated with a red circle.

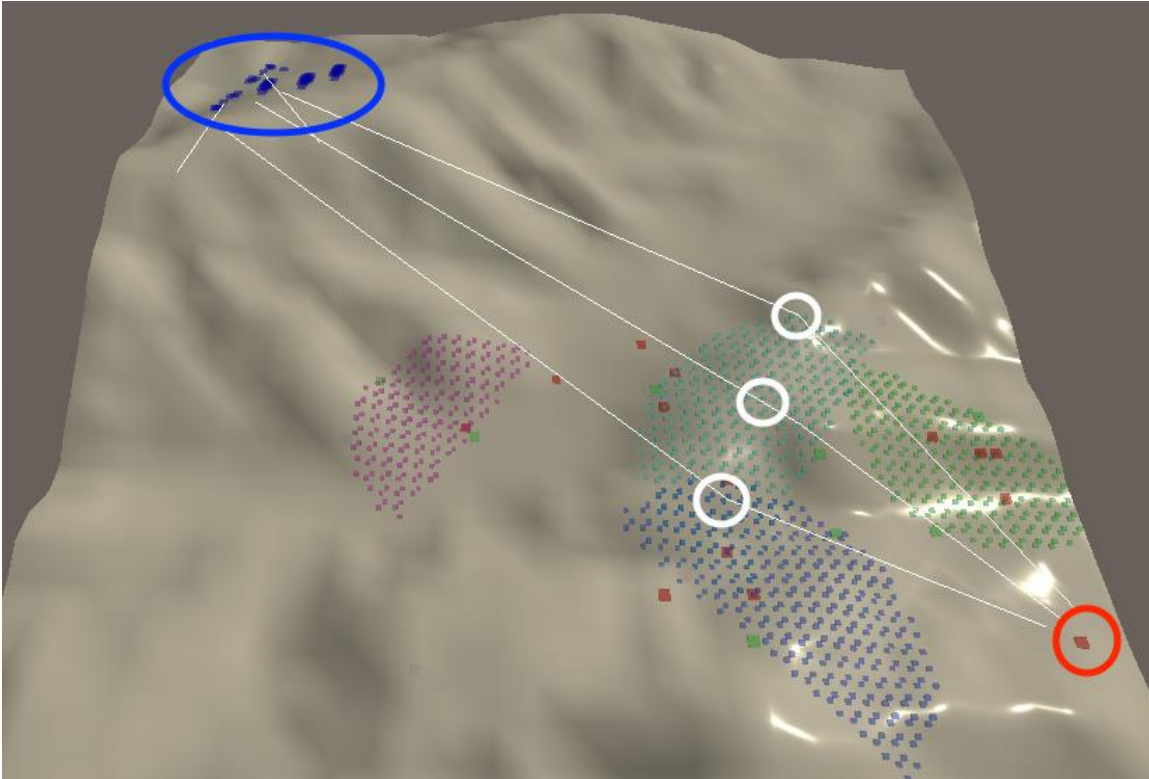


Figure 40. Formation Scoring Evaluation Scenario.

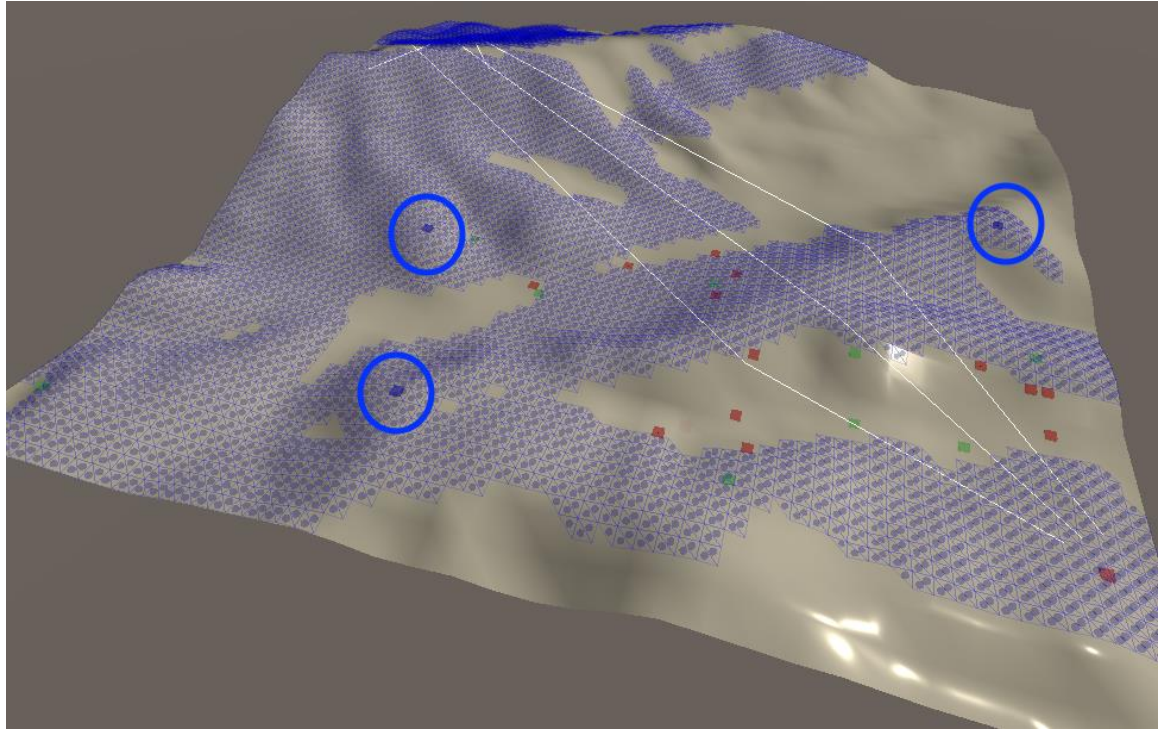
Figure 40 also shows the FOF assignments for red's formation scoring function. These FOF were selected for the following reasons. The OP's FOF (light purple) in the central valley was set to harass the middle of the map and occupy blue's machine gun fire support. Red's fire-teams were assigned to cover a box around the squad leader. Their FOF covered any blue avenue of approach and were designed to take advantage of the second ridge to make blue suppression more difficult.

The desired distances and penalty weights for the formation scoring function are shown in Figure 41.

Targeted Penalty Weight	10000
In Line Of Fire Penalty Weight	0
Preferred Target Weight	0
Field Of Fire Weight	100
Distance From Commander Weight	1
Dispersion Weight	1000
Cohesion Weight	1e+07
▼ Preferred Targets	
Size	0
► Field Of Fire	
Visibility Check	None (Game Object)
► Held Unit Members	
Hold Units	☐
Max Distance From Commander	250
Max Desired Distance	200
Min Desired Distance	100

Figure 41. Formation Score Penalty Weights

As we mentioned earlier, distance values are exaggerated to allow red forces to cover more of the map. This is especially true for the maximum distance from the commander, since the red squad leader is being held to the corner of the map. For this reason, we also reduce the distance from the commander penalty weight. The cohesion weight is set higher to prevent entities from clumping on top of each other. The preferred target weight is zero because only FOF are assigned. Since WXXI does not account for friendly fire, we set the line of fire penalty to zero. This is done to reduce the formation scoring function's dependence on something that would not impact battle outcomes. We set a high cover penalty weight (listed as "Targeted Penalty Weight") to have red favor positions that make blue suppression difficult. For cover calculations, we used a VisibilityGraph created with three representative blue machine guns. This graph is shown in Figure 42.



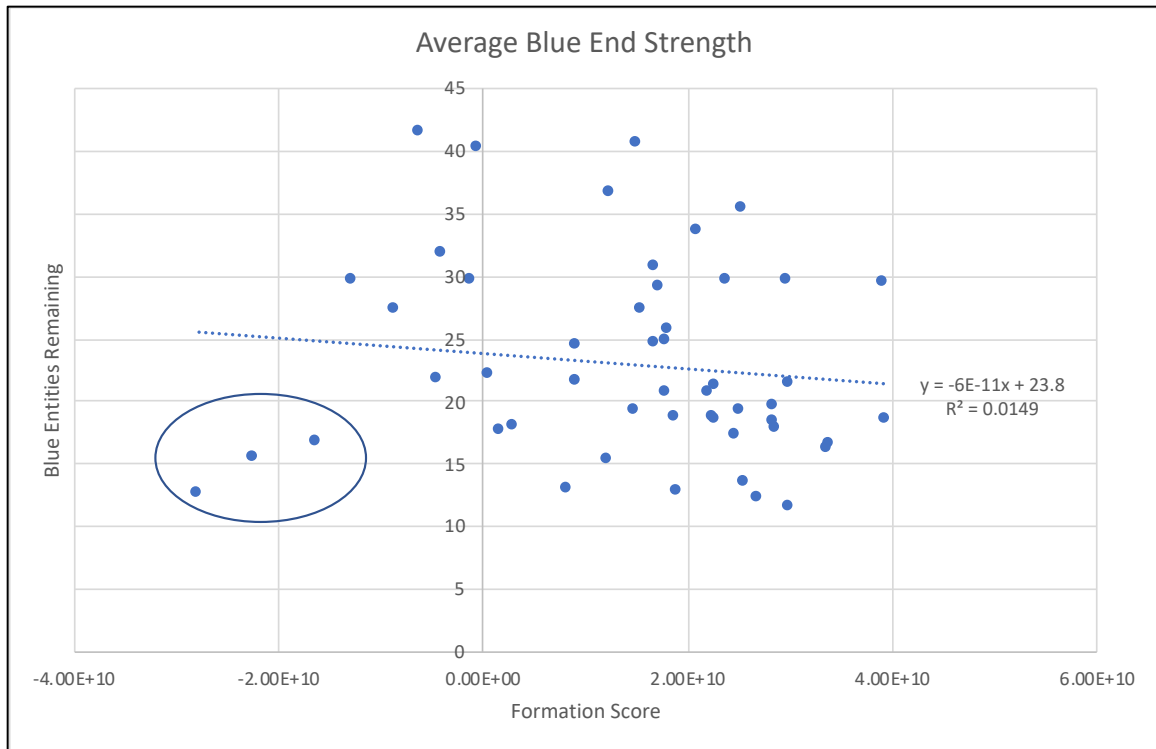
Representative Blue machine guns are highlighted by the blue circles.

Figure 42. Blue VisibilityGraph for Cover Calculations.

2. Results/Analysis

Since WXXI's simulator produces stochastic battle results, we computed averages for blue end strength, red survival rate, mission accomplishment score, and FER for each formation across its 25 battles. We then made a scatterplot for the results and performed a simple linear regression.

When looking at the results for all 50 formations we noticed a fairly weak correlation between formation scores and the four indicators of battle outcome. One of the main causes of this was that three of the worst scoring formation had some of the highest battle results. Figure 43 is a scatterplot of the average blue end strength versus formation scores for all 50 formations. The three low scoring formations are circled in blue.



For this data $r = -0.122$ and the P-Value = 0.398

Figure 43. Scatterplot of Blue End Strength versus. Formation Scores for All Formations.

After visually inspecting these outliers we found that the formations all had instances of extreme cohesion penalties assessed against their formation scores. An example of two occurrences from the lowest scoring formation is shown in Figure 44. While hill climbing can usually move entities out of more extreme penalties, it can become stuck causing the lower score. In the real world, keeping good spacing between unit members prevents an area-of-effect weapon, such as a grenade, from hitting multiple targets. Since there are no area-of-effect weapons in WXXI at this time, the cohesion penalty, while useful for preventing clumping of entities, can cause some useful formations to have unnaturally lower scores. With this in mind, we redid the linear regression without the three outliers.

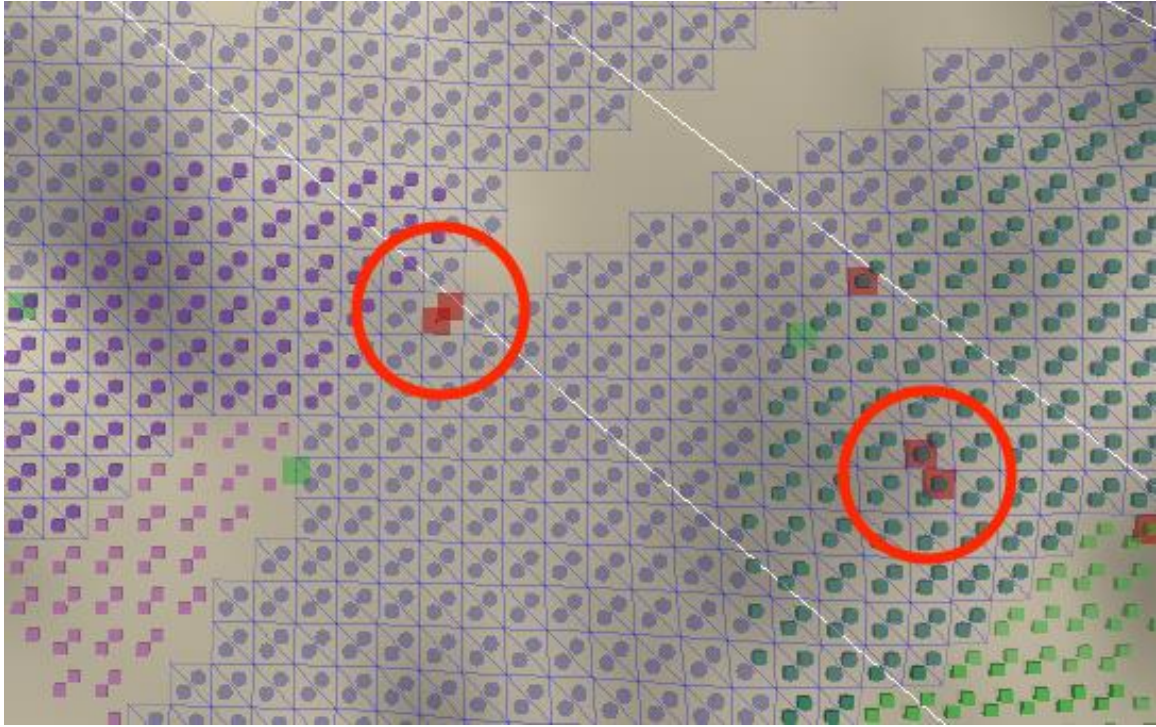
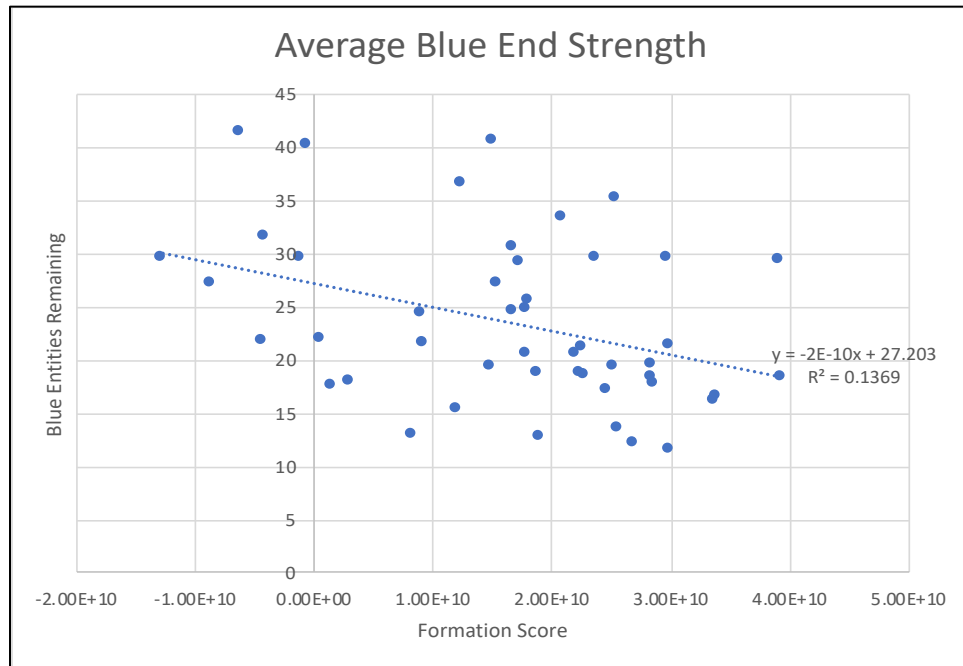


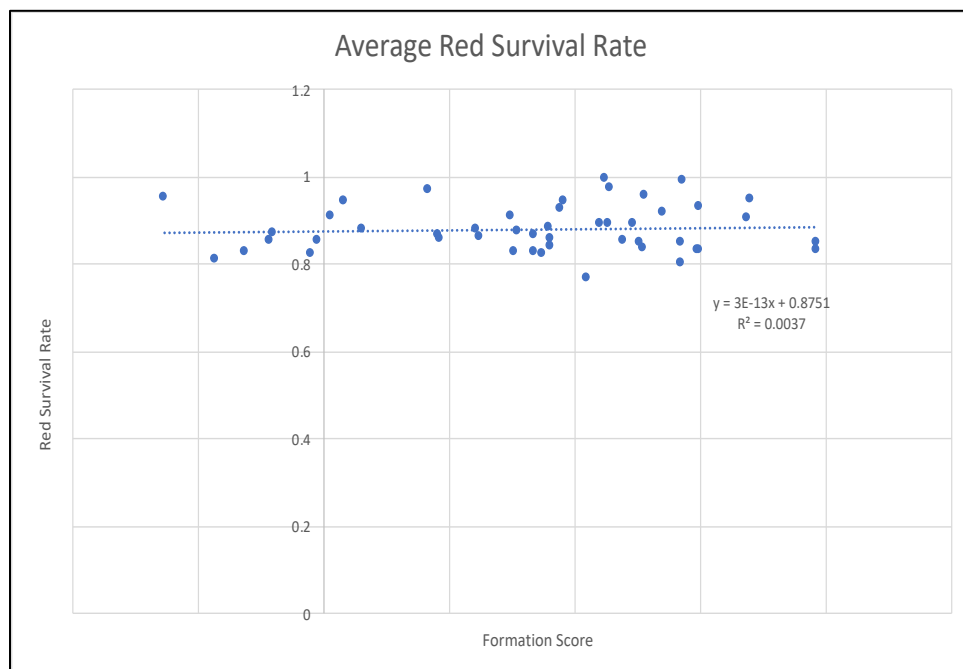
Figure 44. Cohesion Penalty Example.

The scatterplots without the outliers for the four battle indicators are shown in Figures 45–48. In looking at these results, our goal is to assess whether an increase in red's formation score negatively affects blue's battle outcome indicators. For these tests, we will use the criteria $\alpha = 0.05$. For each scatterplot, the best fit line, equation, and R^2 are shown as well.



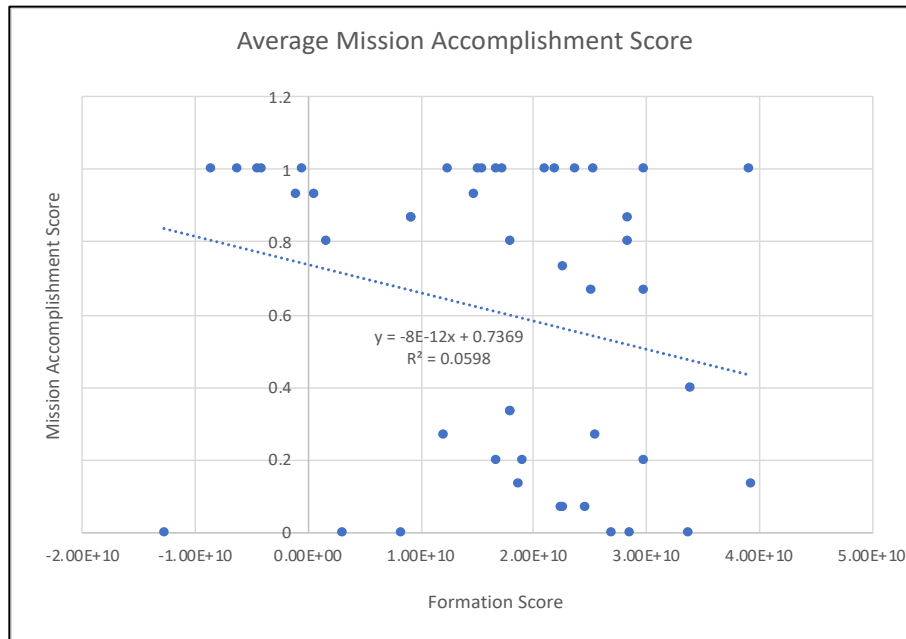
For this data, $r = -0.370$ and the P-Value = 0.010.

Figure 45. Blue End Strength versus Formation Scores (Outliers Removed).



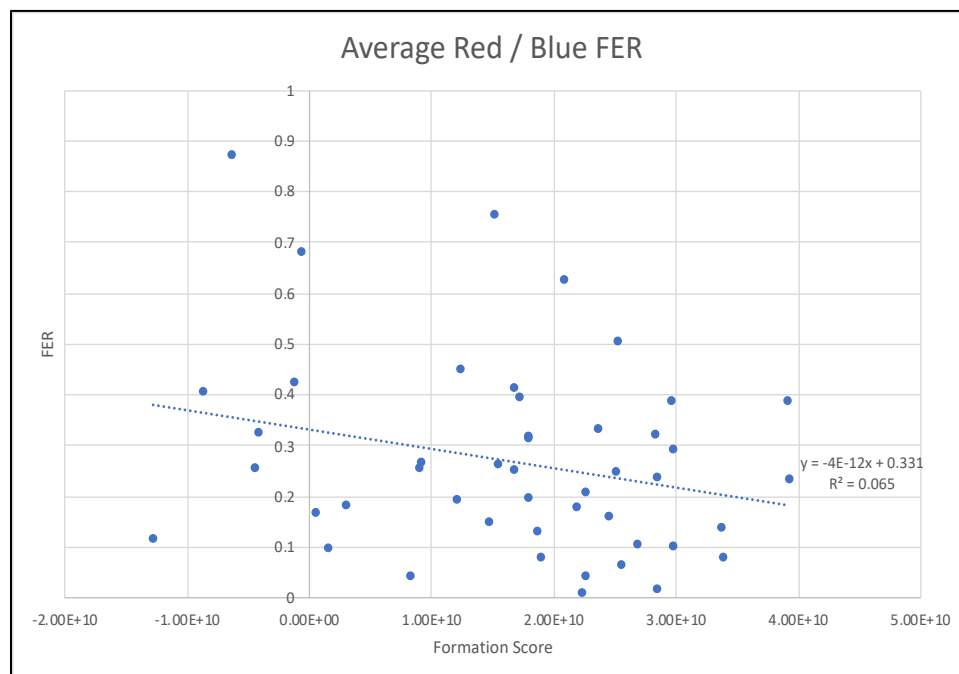
For this data, $r = 0.061$ and the P-Value = 0.686.

Figure 46. Average Red Survival Rate versus Formation Scores (Outliers Removed).



For this data, $r = -0.245$ and the P-Value = 0.098.

Figure 47. Average Mission Accomplishment Score versus Formation Scores (Outliers Removed).



For this data, $r = -0.255$ and the P-Value = 0.0836.

Figure 48. Average FER versus Formation Scores (Outliers Removed).

While only the results for blue end strength showed a statistically significant impact from an increase in formation score, the trends for each of the indicators did show a general decrease in outcomes for blue as formation scores went up. This reflects how some components of the formation scoring functions may not be as important as others, particularly in WXXI. As mentioned earlier, due to the absence of area-of-effect weapons in WXXI, some of the spacing penalties become more relevant to tactical aesthetics, than actual impact to combat performance in the simulator. Conversely, the ability to kill more blue attackers is significantly improved by targeting the right areas and avoiding suppressions. Finally, it is also possible that the defense plan we developed (encoded by the scoring function) had its own weaknesses. It is important to note, that our goal was not to find the perfect defense formation, but to illustrate that by using a defined set of priorities, red could improve its battle performance.

Our interpretation of the results for each specific battle indicator are discussed in the following subsections.

a. *Average Blue End Strength*

As mentioned earlier this indicator showed the strongest correlation to the formation score. Formations that found good cover would force blue's machine guns to have to move across the valley in the middle of the map and over the second ridge. This meant that more of the blue assault advance was done without suppression and the blue machine guns could also be fired upon, increasing the kill count. Since the formation scoring function only used representative blue positions for cover calculations, blue fire support positions that differed from the representative positions could cause lower performance in a formation with a high score.

b. *Average Red Survival Rate*

This indicator showed no correlation to the formation score. Most red casualties, outside of the squad leader were in exchanges with fire-support, leaving the survival rate as more of an indicator of how lucky machine-guns got with WXXI's random hit detection.

c. *Average Mission Accomplishment Score*

While the average mission accomplishment score did not have a statistically significant correlation to the formation score, it did show a general trend of higher formation scores having a lower mission accomplishment score for blue. Preventing mission accomplishment is dependent on being able to eliminate the blue assault element. The correlation of average blue end strength to formation score helps to explain this trend.

d. *Average FER*

Similar to the average mission accomplishment score, the average FER indicator did not have a significant correlation to the formation score, but it also had a favorable trend for red with higher scores. This is consistent with the correlation found in average blue end strength as higher blue casualties would lower the FER, which is better for red.

3. *Summary*

In performing this experiment, we wanted to see how formation scores acted as an indicator for red's outcome in battle. While we did experience some difficulties with how components of our scoring function were applicable in the WXXI simulator, we do feel that given a tactically sound scoring function, it is reasonable to expect improved red performance. Additionally, this type of evaluation would be useful for determining the entropy for a Gibbs distribution of the candidate set. As formation scores have a stronger correlation to improved red performance, it is reasonable to assume that the scoring function being used is a good predictor of the actual red formation. Entropy for the Gibbs distribution should then be decreased to put more weight on the higher scoring formations.

D. *UNCERTAINTY THREAT GRAPH EVALUATION*

For our second experiment, we wanted to test our uncertainty threat graph implementation and see how the blue force would perform given different levels of information about red's positions. Specifically, we wanted to observe the difference between blue planning with partial knowledge of red's forces, an estimation of red's formation (constructed with our Gibbs distribution), and the exact knowledge of all red positions. We collected the same battle outcome indicators as the previous example. While

we expected blue's performance to improve with more (and better) intelligence, we also wanted to see if our estimation of positions could approximate perfect knowledge planning.

1. Scenario Specifics

The initial laydown of forces for this experiment is shown in Figure 49. The starting position for the blue platoon was unchanged, but the routing control points (circled in white) moved closer to the objective (the red squad leader). Unlike the previous experiment, we wanted to give the blue forces more freedom of movement and the ability to avoid encounters if needed.

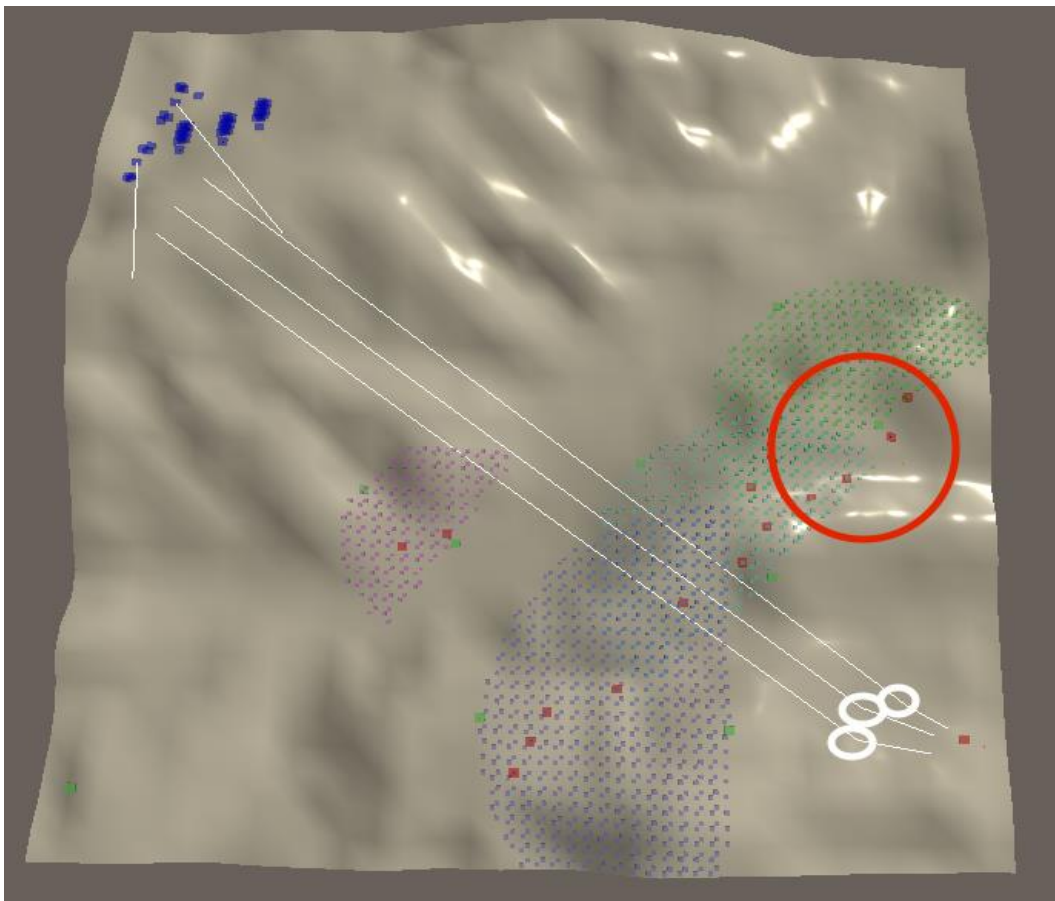


Figure 49. Uncertainty Threat Evaluation Scenario

The red fire-teams have been positioned in a line of defense slightly behind the second ridge. The OP was again placed in the middle valley.

In this scenario, blue was given intelligence that provided them the exact position of the fire-team circled in red in Figure 49. Blue used this intelligence to plan and execute an attack on the red squad leader in two ways. In the first attack, blue planned only for the revealed red forces. This attack simulated blue planning with incomplete information about red's defense. For the second attack, blue planned using an estimation of the threat from possible red positions. This attack demonstrated blue using the expected red tactics to create an uncertainty threat graph for planning. Finally, we conducted one more attack giving blue the exact positions for all of red's forces. This attack represented planning with perfect information and was a comparison point for the different attacks.

While the previous experiment focused on different formations for each battle run, this experiment used constant formations with differing VisibilityGraphs. Three different VisibilityGraphs were used, with 20 battles simulated for each graph. The attack plans generated by each VisibilityGraph were deterministic, but the battle results were stochastic.

In the next sections, we examine the VisibilityGraphs for each planning scenario and the resulting assault actions taken by the blue platoon.

2. Blue Planning for Only Visible Enemies

To simulate planning that accounts only for the observed red fire-team, we created a VisibilityGraph that only included the risk imparted by the entities from that fire-team. This VisibilityGraph for detected enemies is shown in Figure 50.

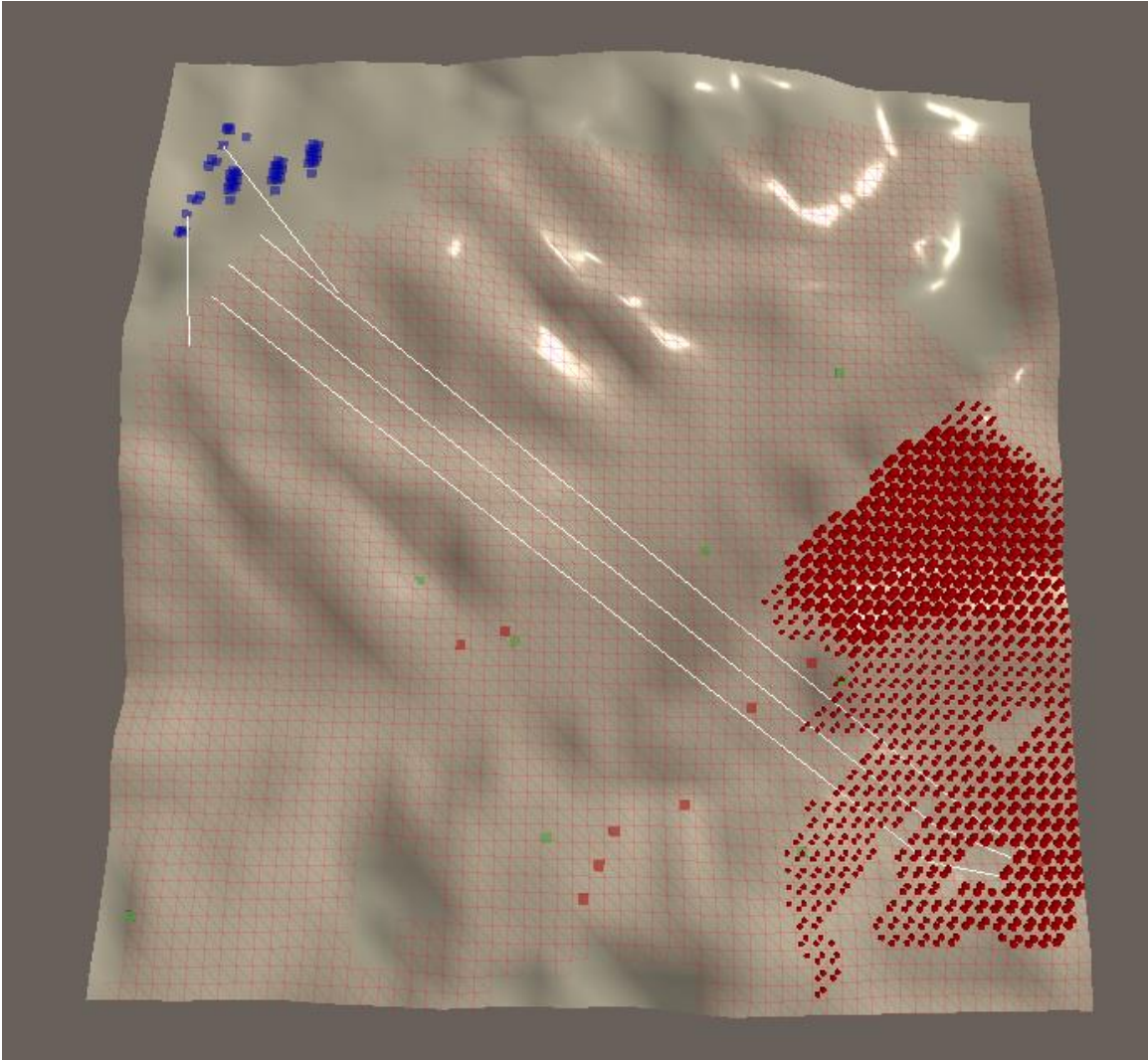
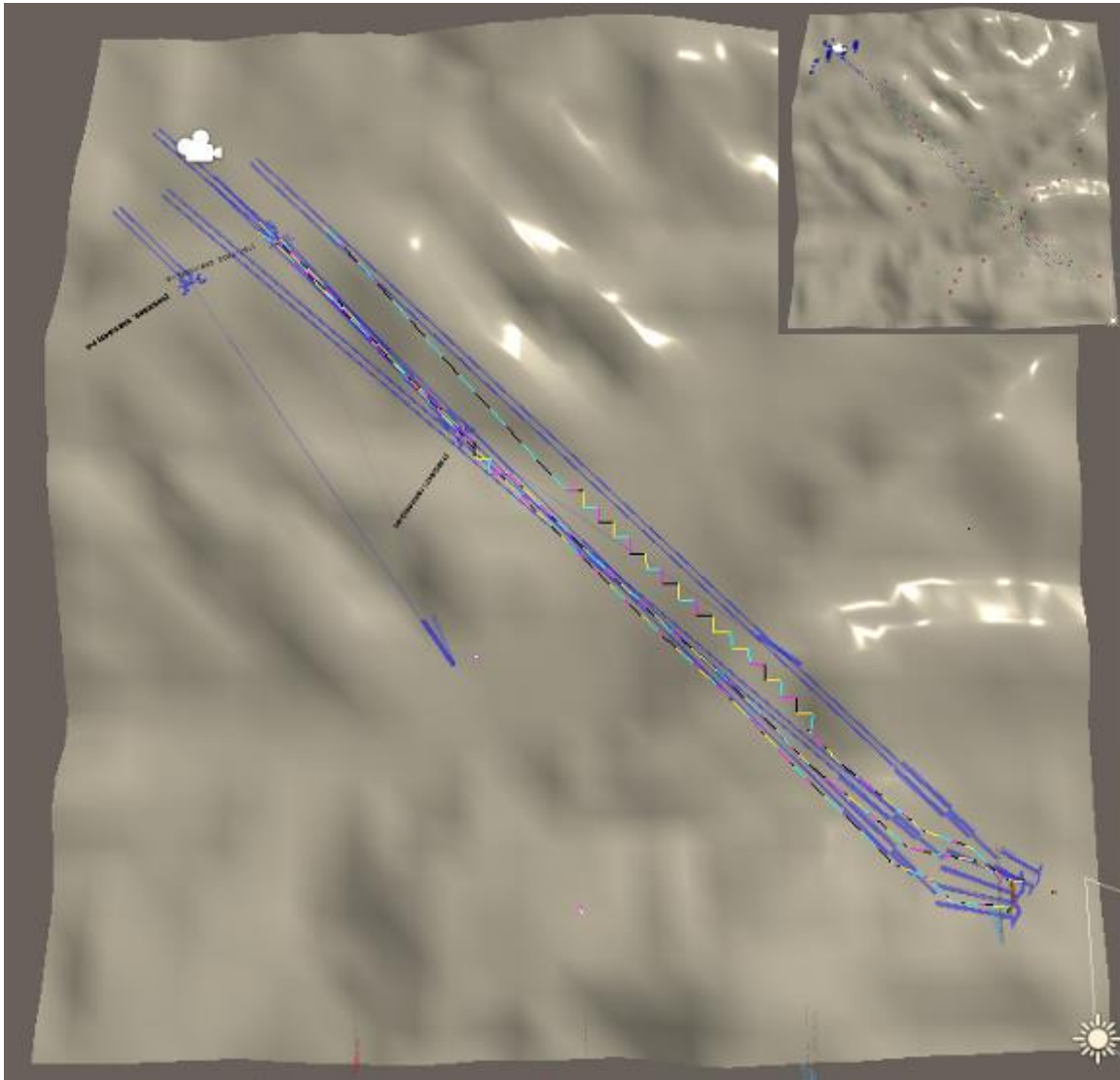


Figure 50. VisibilityGraph For Detected Enemies.

As mentioned in Chapter II, annotations for the entities threatening a node and entities that are targetable from the node, are abstracted from exact entity positions to the more general unit positions. This is why the VisibilityGraph in Figure 50 shows more of the map visible to red than they can actually see. We could not remove these annotations completely, as it would break the fire support planner. We observed that in the absence of perfect information, the fire support elements behaved reasonably and would suppress visible entities near the assaulting force's path.

Without accounting for the rest of the red squad's forces, the blue platoon executes a direct attack at the middle of the red defense line. Their full assault plan is shown in Figure 51.



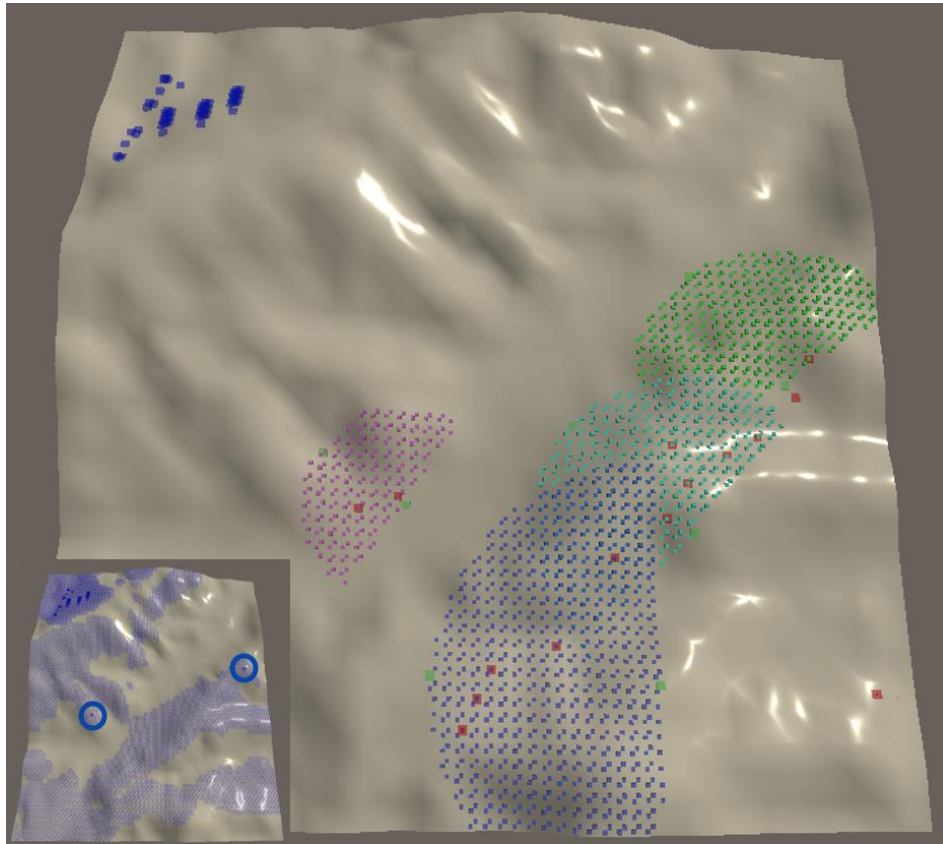
Insert shows assault force pathfinding and red positions.

Figure 51. Attack Plan for Observed Entities.

During their attack blue is able to avoid fires from the detected fire-team and provide suppression to the OP once it is visible. Unfortunately, unsuppressed fire from the other fire-teams takes a heavy toll on the blue forces.

3. Blue Planning with Estimated Risk

In order to simulate planning with an estimation of the threat from red's forces, we first needed to create an uncertainty threat graph. We assume that blue has performed a suitable IPB and identified the correct tactic that red is using (line defense on the ridge). We use this tactic to encode our scoring function. FOF assignments and the cover calculation graph are shown in Figure 52. We kept the desired distances and penalty weights the same as the previous experiment (seen in Figure 41).



Cover calculation VisibilityGraph with two representative blue machine gunners is in the bottom left insert.

Figure 52. FOF Assignments and Cover Calculation VisibilityGraph.

We used this scoring function and the hill climbing method to create a candidate set of 50 formations. To calculate the Gibbs distribution for the candidate set we used an entropy value of 0.70. We felt that this value allowed the resulting uncertainty threat graph

to incorporate elements from most of the formations, while still given some weight to the formation scores. With entropy at 0.70, the highest-scoring formation had a probability of 0.22. Figure 53 shows the resulting uncertainty threat graph. The right side of the graph has features that match the VisibilityGraph for detected enemies (Figure 50). This is expected as the revealed fire-team was not included in the hill climbing and its entities' positions were kept at their observed value for every formation in the candidate set.

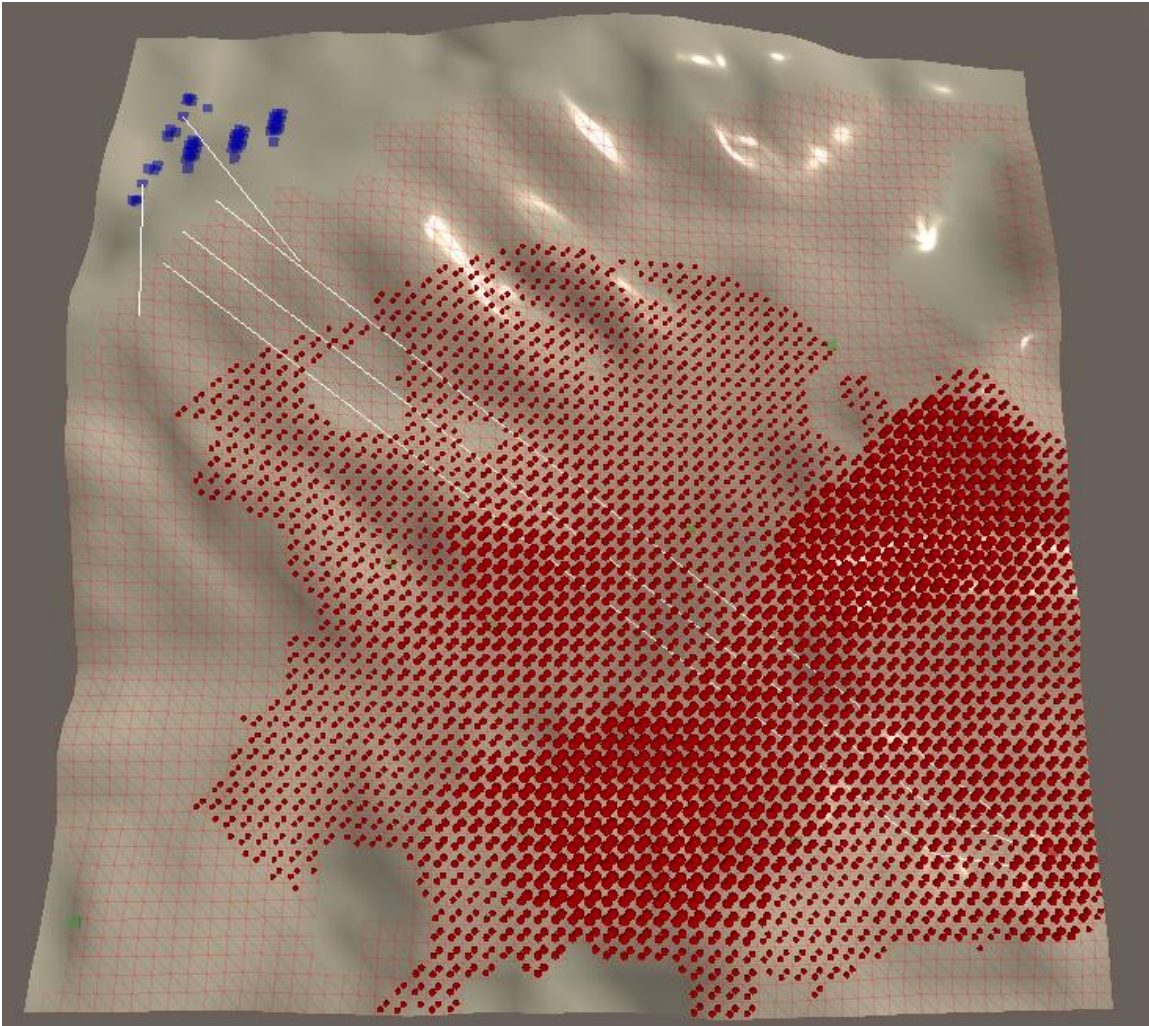
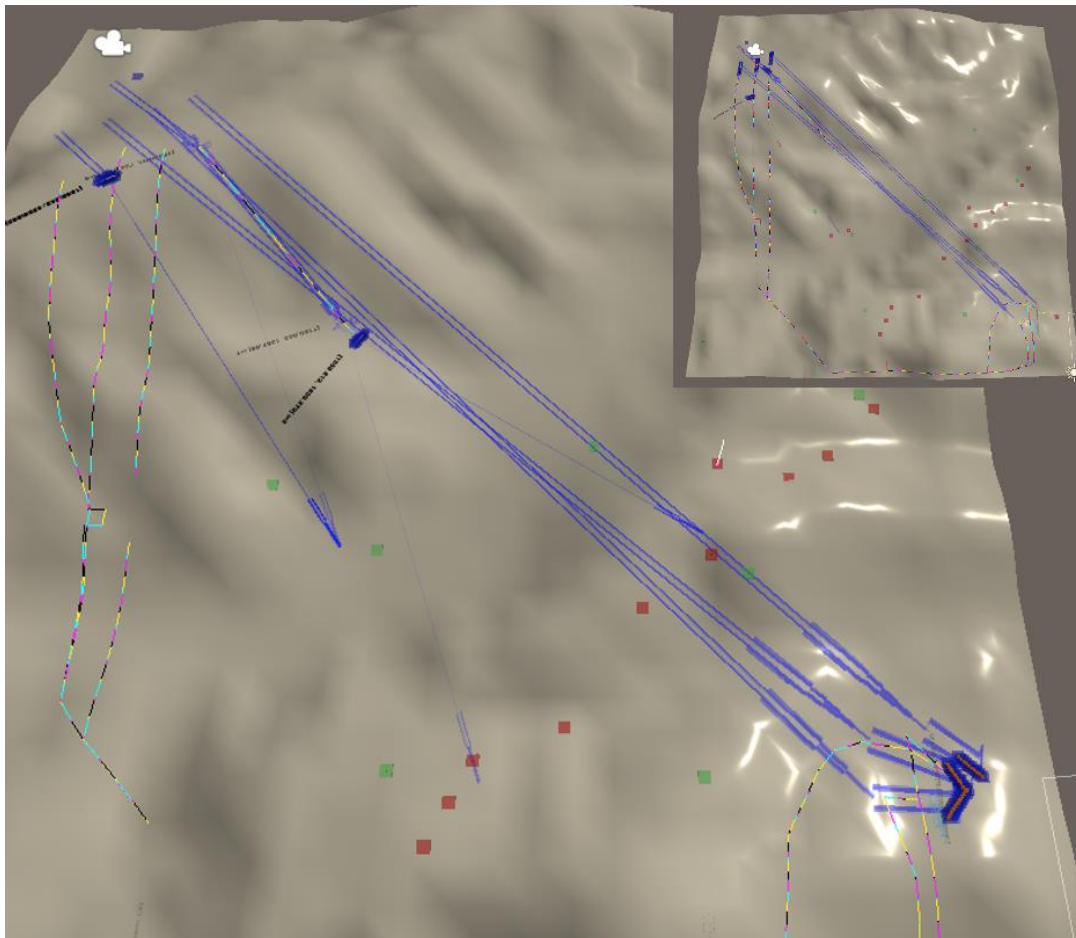


Figure 53. Uncertainty Threat Graph form 50 Formations with Entropy = 0.70.

Node annotation is again abstracted to unit position, which is unchanged from the Planning for Observed Entities scenario.

When using the uncertainty threat graph for planning blue is able to identify a less risky path around the red defense line. Additionally, blue's machine gun squads provide effective suppressive fire to the OP and red's left flank as the assault element moves past them. While the fire-support becomes less effective once the assault element goes behind the second ridge, blue's casualties are greatly reduced when compared to the previous battle. Blue's full fire support plan and pathfinding are shown in Figure 54.



Insert shows assault force pathfinding.

Figure 54. Plan of Attack created with Uncertainty Threat Graph.

4. Blue Planning with Perfect Information

For our final battle scenario, we allowed the fire support planner to run with full information about red's positions and non-abstracted node annotations. The VisibilityGraph for red's actual formation, and blue's assault force pathfinding are depicted in Figure 55.

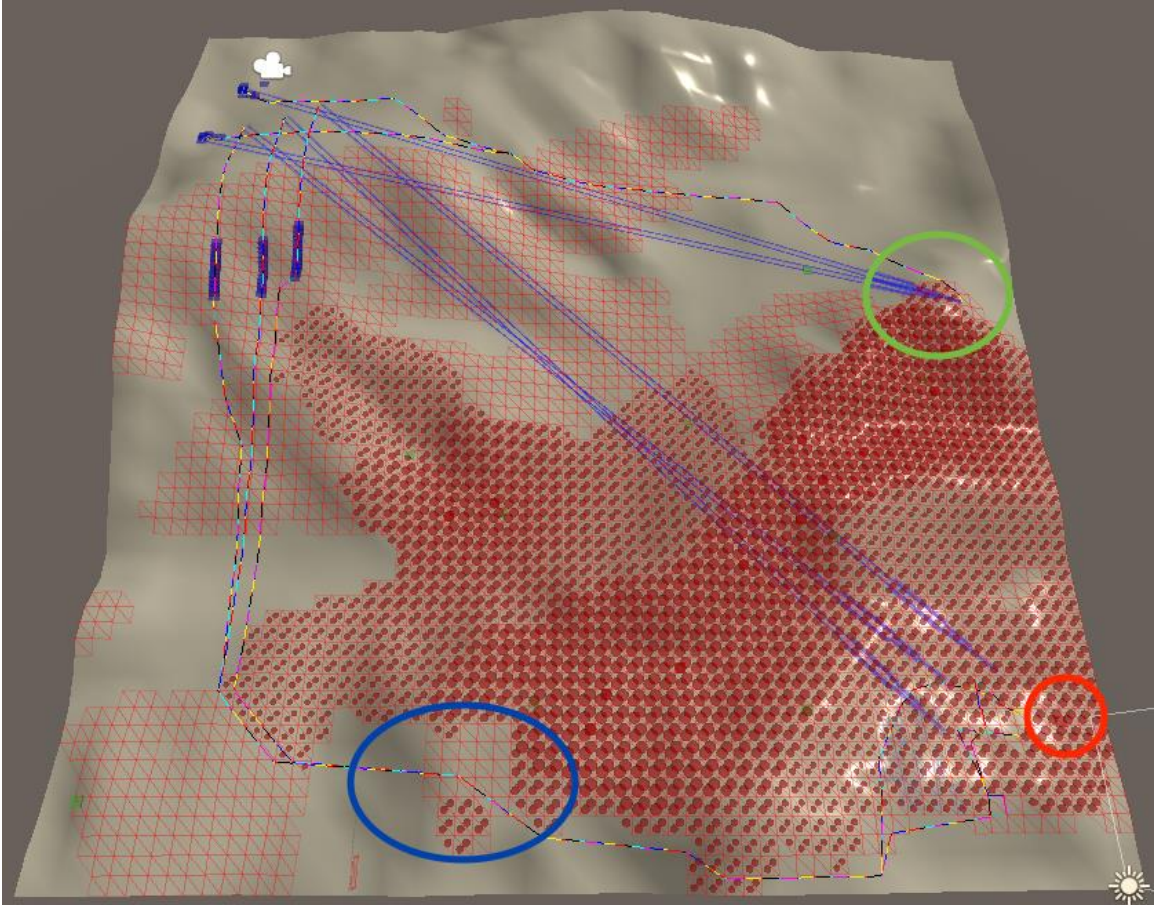


Figure 55. Perfect Information VisibilityGraph with Blue Pathfinding.

While the plan is similar to the one generated with the uncertainty threat graph, its pathfinding is much more precise. Blue's assault force almost completely avoids red's ability to target it. This level of precision would be unrealistic in an actual battle. The blue circle in Figure 55 gives an example of the attacking force "threading the needle" as it flanks the red defense line.

The uncertainty threat graph plan and the perfect information plan differ the most with regards to fire support. Since blue knows that its assault squads will be minimally targeted as they move around red's flank, the machine gun squads move to the opposite side of the red's defense (Figure 55, green circle). In this position, they are just barely out of range from red's rifles, but they can completely suppress the red squad leader in the bottom right corner (Figure 55, red circle).

5. Battle Results and Analysis

Indicator results for the 20 battles fought with each different graph are shown in Table 2.

Table 2. Battle Indicator Results for Uncertainty Threat Graph Evaluation.

Scenario	Attacker (Blue) End Strength		Defender (Red) Survival Rate		Defender/Attacker FER		Mission Accomplishment Score	
	μ	Std. Err	μ	Std. Err	μ	Std. Err	μ	Std. Err
Detected Enemies Graph	32.50	1.00	0.79	0.02	0.56	0.06	0.70	0.11
Uncertainty Risk Graph	47.50	0.63	0.78	0.01	2.28	0.32	1.00	0.00
Perfect Information Graph	51.35	0.40	0.93	0.00	1.92	0.28	1.00	0.00

As expected, results generally improved as blue gained more information about red's formation. The following sections provide a more detailed analysis of the results for each indicator.

a. Average Blue End Strength

The indicator for average blue end strength behaved in a predictable manner. Box plots for the results for each scenario are shown in Figure 56. While the average survival was higher when planning with perfect knowledge, planning with the uncertainty threat graph did produce some battles that had comparable performance.

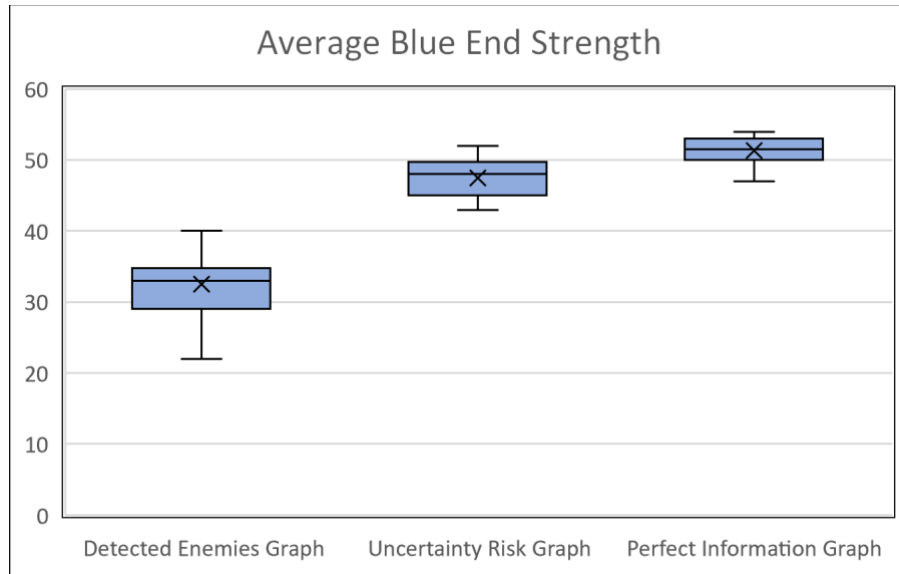


Figure 56. Box Plots of Average Blue End Strength.

b. Average Red Survival Rate

The indicator for average red survival rate was the one area that planning with perfect information produced the worst results. Box plots for average red survival rate are shown in Figure 57.

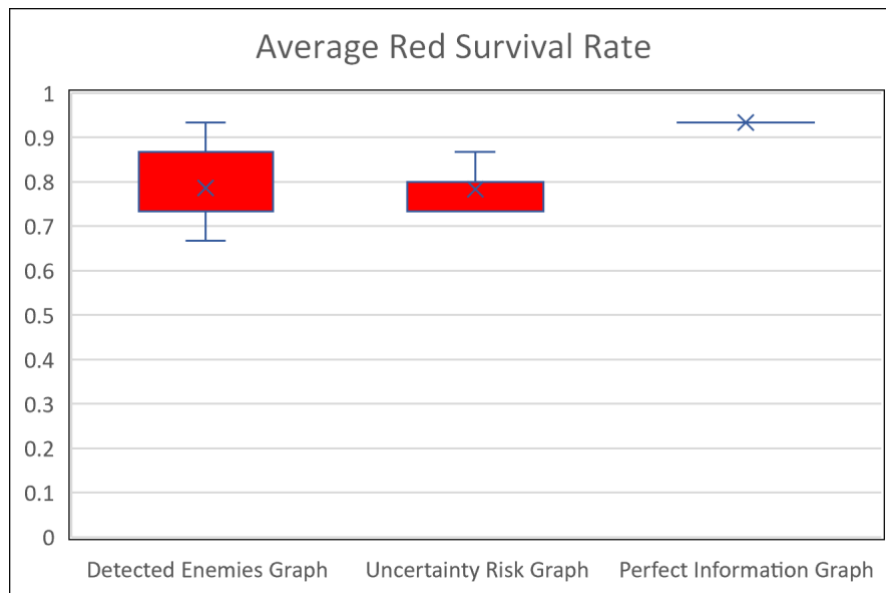


Figure 57. Box Plots of Average Red Survival Rate.

As in the previous experiment scenario, red casualties, outside of the squad leader were caused by the fire support element. For battles fought without perfect information, the machine gun squads would occasionally wipeout the red OP entities and get kills on other suppression targets. The plan generated with perfect information only suppressed the red squad leader, so it did not have any other kills.

c. Average Fractional Exchange Ratio

The uncertainty risk graph had a higher average FER than the perfect information graph. This is most likely caused by the perfect information graph only killing the red squad leader in all battle runs. Box plots for average FER are shown in Figure 58.

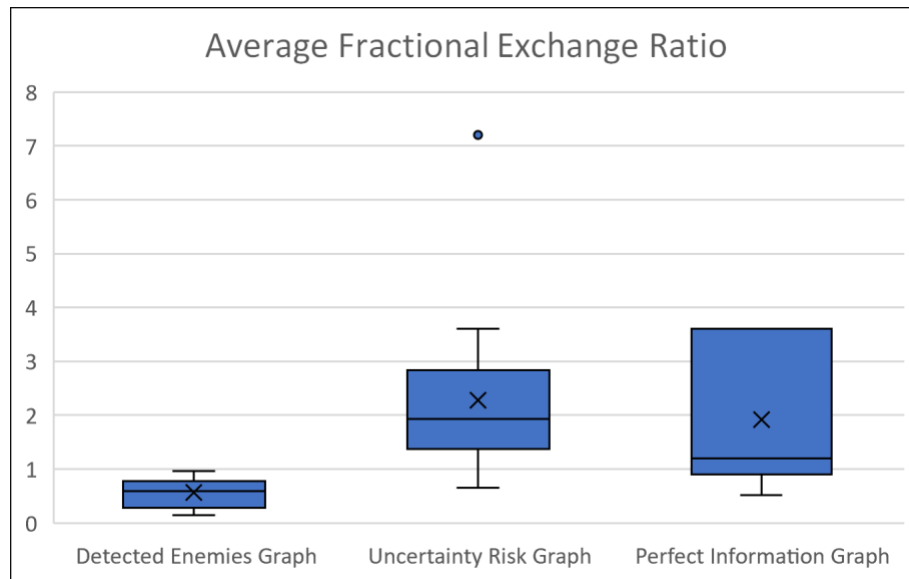


Figure 58. Box Plots of Average FER.

While the uncertainty risk graph did have one run with a FER of 7.2, the perfect information graph had a battle run with no blue casualties. As the calculating the FER for this run is impossible, it is not represented in the box plot.

d. Average Mission Accomplishment Score

The uncertainty risk graph and the perfect information graph killed the red squad leader in all their runs for an average mission accomplishment score of 1. The detected

enemies VisibilityGraph did not always kill the squad leader and had a mission accomplishment score of 0.7.

6. Summary

The results of this experiment demonstrate that a proper scoring function can generate an uncertainty threat graph that performs comparably to a perfect information VisibilityGraph. We found that both the complete information graph and the uncertainty threat graph produced similar maneuver plans, but much different fire support plans. While the perfect information plan produced the best survival results, its flawless positioning and movement was actually detrimental to some of the other indicators.

Even though the “detected enemies only” VisibilityGraph performed predictably worse than the other graphs, it does serve as an example of implementing fog-of-war effects in WXXI. Utilizing an incomplete VisibilityGraph effectively blinded blue’s forces to the other units when conducting pathfinding. Additionally, limiting information permitted other desired features in the simulation. By giving blue imperfect information, we forced more dynamic behavior from their supporting fire squads and allowed red defenders to set ambushes. This idea is explored further in the next section.

E. BLUE PLANNING WITH FOG-OF-WAR

For our final test we used multiple incomplete VisibilityGraph’s to simulate blue planning with fog-of-war effects. We did not collect any quantitative data for this evaluation but focused on the pathfinding results from the maneuver planner. While this was not a direct test of prediction methods, it did demonstrate a path to an added capability for planning movement to contact operations.

1. Deliberate Attack Verse Movement to Contact

Harder justifies his fire support planner’s use of perfect information by limiting its focus to deliberate attack operations (2017) These attacks require “detailed reconnaissance to pinpoint objectives, [and] enemy positions” (U.S. Marine Corps 2014, 6–9). In the absence of this reconnaissance, commanders often execute a movement to contact in order to collect information about the threat environment and support following actions. These

operations entail direct aggressive movement to a likely enemy position. Once contact is made, the attacking forces can engage with the enemy, bypass them, or withdraw (U.S. Marine Corps 2014, 6–6). Adding fog-of-war to the WXXI combat simulation changes the doctrinal situation for the fire support planner from a deliberate attack operation to a movement to contact. With this change in doctrinal situation, we expected that the fire support planner outputs when planning with limited information would also reflect attributes found in movement to contact operations, particularly with regards to pathfinding.

2. Scenario Specifics

For this scenario, we use the same ridge line defense by red that was used in the previous experiment. The blue objective remains to kill the red squad leader and control points are similar to the second experiment. We start the blue platoon with a blank VisibilityGraph and abstracted annotations, simulating no knowledge about the red positions. Figure 59 shows the starting conditions for the battle.

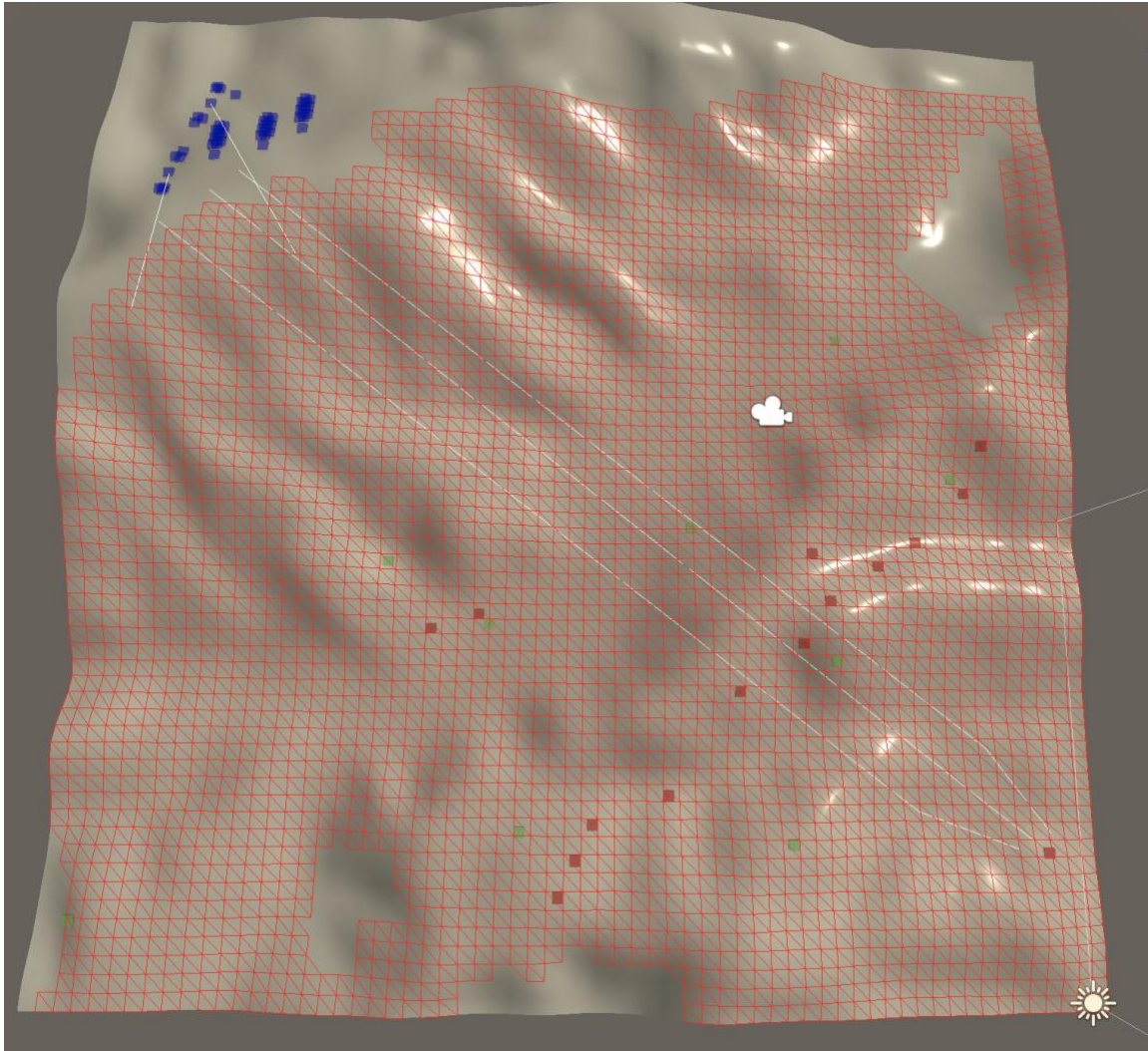


Figure 59. Fog-of-War Evaluation Starting Conditions With Zeroed VisibilityGraph.

During the battle we manually control the update of blue's VisibilityGraph with the following actions. First, we pause the simulation when blue has visual contact with a red unit and save the current position of the attacking platoon. Visual contact is established by seeing an exchange of fire or by a separate VisibilityGraph with the blue platoon set as the threatening unit (similar to how cover is calculated in the formation scoring function). The VisibilityGraph can be rescanned during the battle and it updates the nodes visible to the blue force as they move across the terrain. If blue is able to see a node occupied by a red defender, then we assume that the entity has been detected. We then stop the simulation

and calculate a new detected enemies VisibilityGraph for all of the visible red entities. We restart the simulator with the blue platoon at its saved position and allow the planner to use the updated detected enemies VisibilityGraph for planning. We decided to limit the changes made when restarting the scene, so we left the objective and routes constant for the blue platoon. Since we did not change the routing or control points for the blue maneuver units when we updated the VisibilityGraph, we only expected to see blue's assault squads bypass or withdraw from contacted red forces. Automation of the above steps and objective reassignment are discussed as possible future work in Chapter VI.

3. The Fog-of-War Engagement

Without exact knowledge of red's formation, the blue platoon heads directly for their objective as seen in Figure 60. This indicative of the assertive actions seen in movement to contact operations and a stark contrast to the circuitous routes seen when blue has exact (or even estimated) knowledge of red positioning.

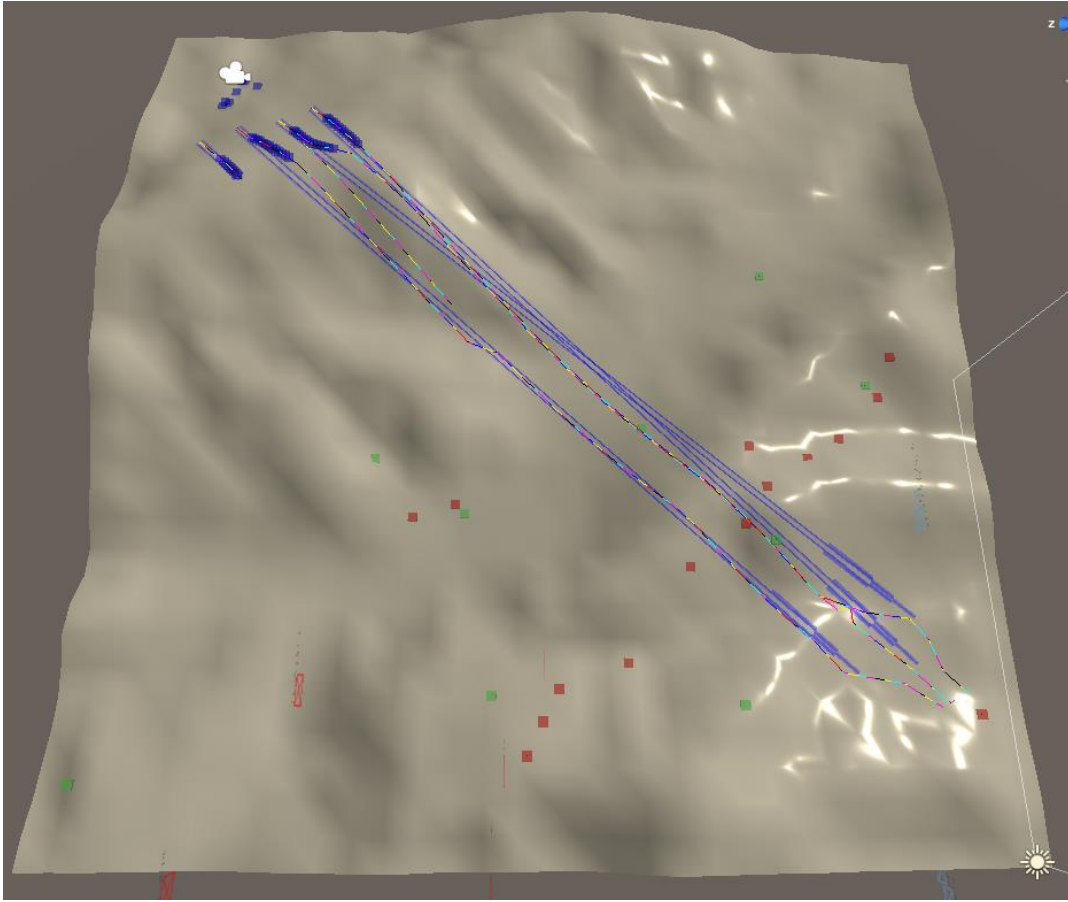
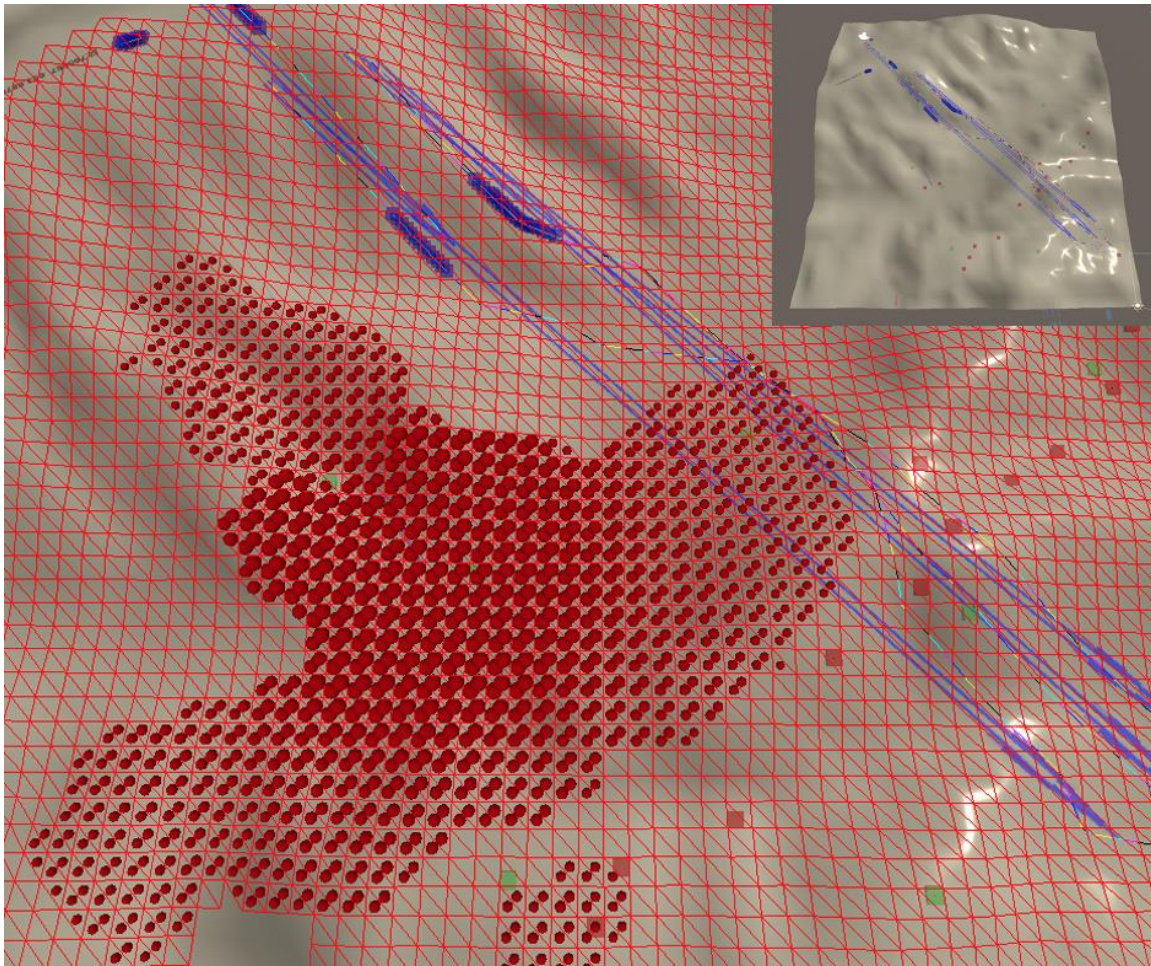


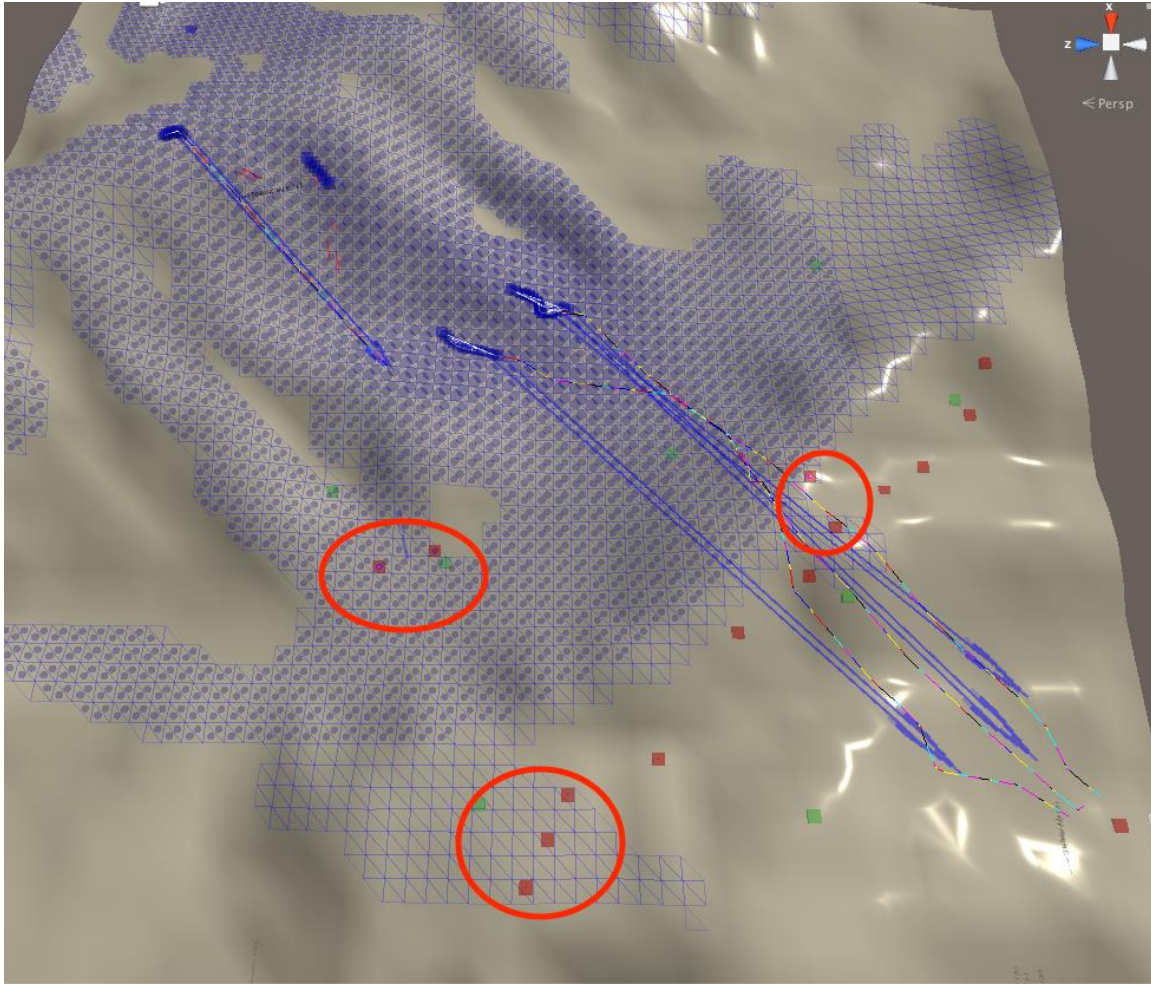
Figure 60. Initial Blue Movement Plan in Fog-of-War Scenario.

Blue first make's contact with red's OP unit. The blue machine guns suppress the OP and the maneuver element adjusts its course to avoid the threat. The updated detected enemies VisibilityGraph and pathfinding are shown in Figure 61.



Top right insert shows full maneuver plan and fire support and fire support on red's OP.
Figure 61. Updated Detected Enemies VisibilityGraph and Pathfinding.

Upon moving further into the valley, blue gains sight of five more red defenders as depicted in the VisibilityGraph shown in Figure 62.



Detected entities are circled in red.

Figure 62. Blue Detects More of Red's Entities.

With this new information, the detected enemies VisibilityGraph is updated (shown in Figure 63) and the blue assault force again elects to bypass the visible defenders. Blue's new path is shown in Figure 64. Blue did not detect any other red entities until after the assault team rounded the ridge and started to proceed towards the squad leader. At this point, updates to the detected enemies VisibilityGraph did not significantly alter blue's actions.

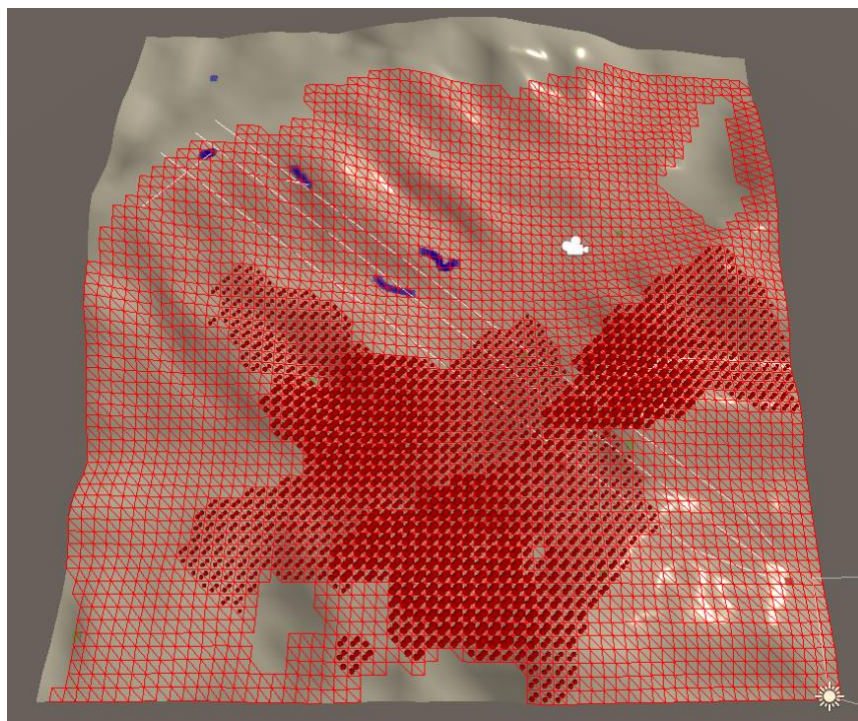


Figure 63. Second Update to Detected Enemies VisibilityGraph.

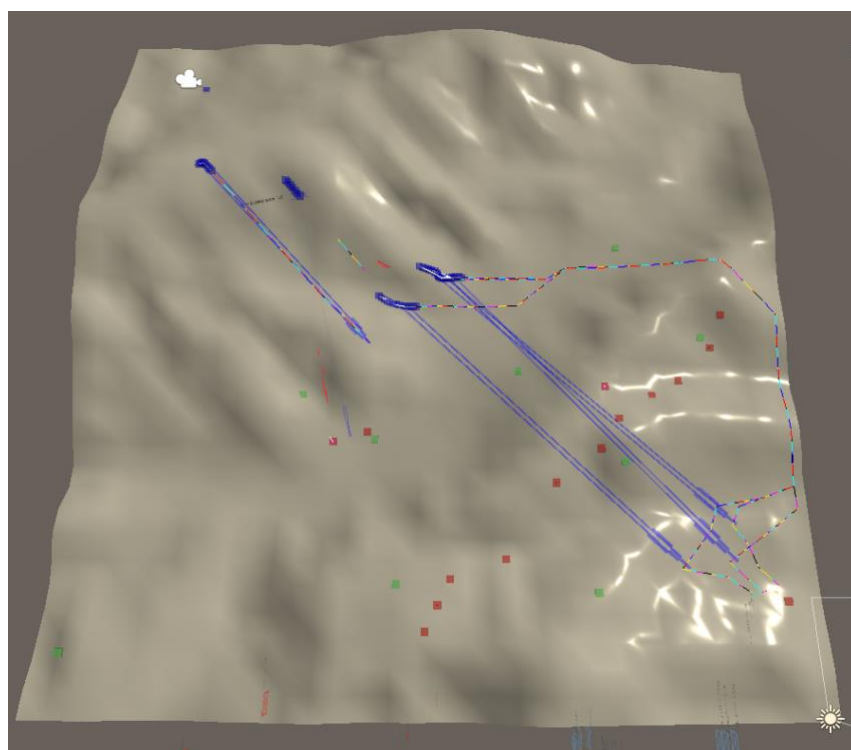


Figure 64. Updated Pathfinding.

4. Updating with Estimated Risk

While the detected enemies VisibilityGraph was able to guide the assault force around the red defense line, it could not account for the defenders hidden behind the ridge until blue was already committed on a direct path to the squad leader. We can improve blue's decision making by creating an uncertainty threat graph using the detected red entities seen in Figure 62.

We used the same formation scoring function from the planning with estimated risk scenario in section 5.D.3. We created a 20-formation candidate set, holding the positions of the detected entities constant for all formations. With detected entities in multiple red units, we had a higher confidence in the formation score results and used an entropy value of 0.2. The resulting uncertainty threat graph is shown in Figure 65.

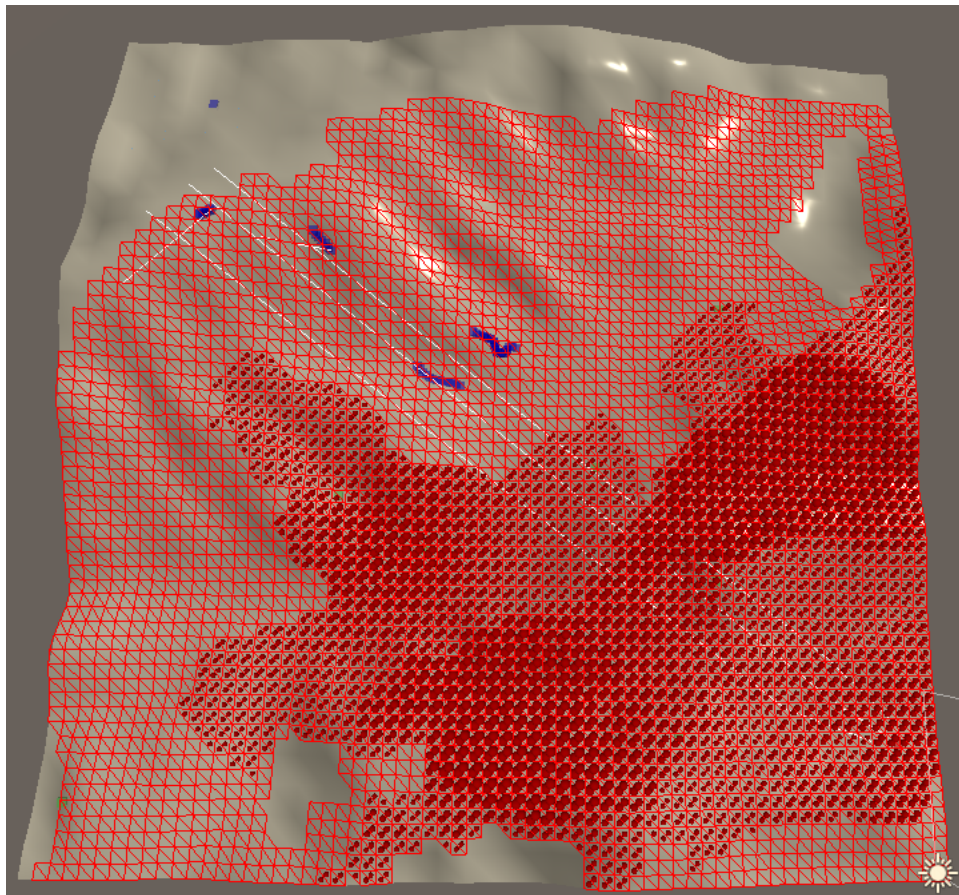


Figure 65. Fog-of-War Uncertainty Threat Graph.

With this estimate of the full red defense, the assault force initially withdraws, then bypasses the red forces on the opposite flank. Updated pathfinding is displayed in Figure 66.

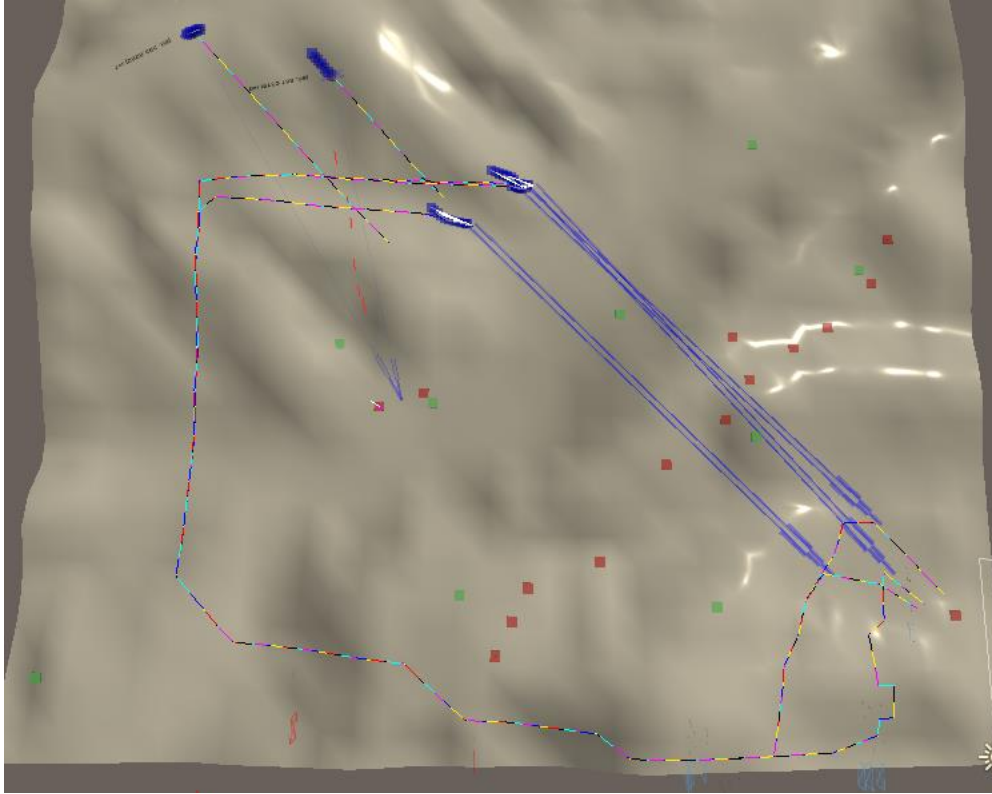


Figure 66. Updated Pathfinding with Uncertainty Threat Graph.

5. Fog-of-War Conclusions

This test illustrated how accounting for fog-of-war effects in a combat simulation can improve agent behavior and add functionality. Without any changes to the actual encoding of blue agents, we could move from a deliberate attack operational planner to a simulated movement to contact, by simply by updating the VisibilityGraphs. In the absence of perfect information, the blue assault force was able to make direct and aggressive movements towards its objective. When faced with updated information, whether actual or estimated, the blue assault force was able to withdraw from or bypass the detected enemy. While engagements with detected enemies was not shown (since this would have required a reassignment of the objective point) it could be added. Severson's module has already

demonstrated the ability to update routing and objective assignments automatically in WXXI.

F. SUMMARY

This chapter showed how we evaluated the functionality of our implementations and how their outputs could be applied in the combat simulator. We created a simple battle scenario and used it to test our formation scoring function and to compare our uncertainty threat graph performance against only observed data and perfect information. Finally, we used our ability to manipulate VisibilityGraphs to add fog-of-war to the combat simulator and showed how this can be used to replicate a movement to contact operation.

VI. CONCLUSIONS

In starting this research, we were motivated to show the importance of fog-of-war effects in combat simulation and how to enable AI decisions when given less than perfect information. We saw the potential benefit that AI approaches could add to military simulations with regard to decision support, planning automation, and wargaming. These benefits depend on realistic AI behaviors. Behaviors that make use of information that is usually unavailable to human combatants are not realistic.

We looked at products from the IPB and how to translate the tactical intelligence it produced into a combat simulation environment. We found that with a navigational graph of the operational terrain and formation geometric data, we could represent adversary templates through node annotations. We used this to build our formation scoring function which provided a way to compare and evaluate possible formations against strategic priorities. We then used random placement and hill climbing to find formations that maximized the formation score and used that for our prediction for likely enemy positions.

To account for hill climbing's weakness to local maximums we used multiple iterations to produce a candidate set of formations that we could then evaluate based on their individual scores. We applied Gibbs work with the canonical ensemble to create a distribution for the candidate set, with probability for each formation dependent on its formation score and the entropy of the system. We used entropy to act as the amount of "faith" that the commander had in the available intelligence (and by that extension the formation scoring function). With this distribution, we created a weighted estimate of the risk implications from all possible formations in the candidate set. We called this estimate the uncertainty threat graph.

We also applied canonical ensemble equations to evaluate probably enemy tactics given a known set of enemy positions. We compared how the different formation scoring functions evaluated the know positions and then calculated a probability for each possible function. While our original distribution compared multiple formations against a single

encoded tactic, plan recognition evaluated many scoring functions against an exact formation.

We were able to implement our methods using the WXXI environment and Unity3D game engine. After checking the functionality of our implementation, we conducted three different evaluations, each testing a different component or use-case of our application. From the qualitative and quantitative data collected we were able to draw several conclusions.

In the next sections we outline our research conclusions and contributions, as well as, identify areas for future work.

A. RESEARCH CONCLUSIONS AND CONTRIBUTIONS

1. Enhancing Combat Simulation by Adding Fog-of-War

We found that by adding fog-of-war to the combat simulator we were able to have improved wargaming and more flexible scenario development. In our evaluation of the formation scoring function (Chapter V Section C) we had to overcome blue's advantage of knowing red's formation by forced routing to specific points on the map. On subsequent evaluations we were able to use less structured encounters by removing blue's perfect information. We could lead blue into an ambush using an incomplete detected enemy VisibilityGraph or avoid threats with an estimated uncertainty threat graph. While the blue platoon had better survivability with perfect information (as would be expected), when uncertain about red's position, the fire support element engaged more of the red entities. Additionally, the units they engaged more closely followed the pathfinding results of the assaulting element.

Furthermore, adding fog-of-war increased the doctrinal range of operations that could be simulated in WXXI. The original fire support planner assumed a deliberate attack and used accurate enemy fire positions for optimal performance. In this configuration it becomes difficult to simulate a movement to contact operation, since blue is able to easily avoid red's forces. We were able to implement a simplified movement to contact through restricted VisibilityGraphs, and without significant changes to the pathfinding implementation or routing orders.

2. The Gibb Distribution and Aggregated Risk Estimate

The application of the canonical ensemble equations to construct the Gibbs distribution is a novel way to evaluate a set of possible formations and aggregate the risk implications of them. We use it to overcome some of the weaknesses of hill-climbing in our method, but it is applicable for any set of formations with an adequate evaluation procedure. Similarly, our method for plan recognition is applicable for evaluations outside of a combat simulation and presents another innovative application of the canonical ensemble equation.

While pinpointing the exact right position for every red entity was difficult, we could reliably forecast the risk implications of the actual laydown of red forces with the uncertainty risk graph. The graph may not improve targeting, but it is useful for pathfinding. This is particularly true with regards to avoiding enemy encounters. Additionally, understanding risk implication can also be applied to asset allocation and employment. We find this to be reasonable because we are not trying to provide blue with the exact red formation, but rather model the risk assessment a blue commander would make with intelligence support in a fog-of-war situation.

3. Encoding the IPB as Formation Score

Our system is distinguished from current techniques in that it seeks to incorporate intelligence products developed in the IPB to make a prediction that is consistent with expected enemy tactics in a given environment. Our method for the evaluation of the enemy squad formation presents a novel extension of tactical waypoint annotations (Straatman et al. 2005) and tactical fireteam movement (Darken et al. 2010).

Reviewing previous work with video games, we found that AI decision-making did not rely on encoding established doctrine or intelligence. Instead, they depended on training across a library of game logs. While these methods produced interesting results, it would be unlikely for a combat planner to have a collection of a battle logs fought on their exact battlespace or the time to retrain the AI against new evolving enemy tactics. Conversely, adversarial templates and other intelligence products are readily understood by combat planners and can be easily added, edited, and visualized in a CSE.

B. FUTURE WORK

WXXI was integral to the development and evaluation of our methods. General improvement to its combat simulator and further tactical considerations for formation scoring function are natural areas for further exploration. Our implementation of the Predictor and PlanPredictor classes in WXXI can be found as branch repository of the Wombat XXI project on the NPS GitLab repository at <https://gitlab.nps.edu/tecom/wombatxxi/-/tree/KJMaroon1>. The modularity of Unity3D and WXXI in particular make upgrades and improvements fairly uncomplicated.

While the hill climbing method worked to provide improved formations from random positions, it would be interesting to investigate other methods of finding maximizing formations. A genetic algorithm or simulated annealing method for formation generation may provide a better candidate set of formations. However, added complexity to the maximizing algorithm may increase the time it takes to generate a candidate set.

Harder (2017) and Severson (2019) both noted the increasing computational load for larger force structures and terrain size. Our current implantation on the Cayucos terrain with 8192 nodes and a 15-entity red force takes about 9.5 seconds to generate each formation in a candidate set. This is accomplished with an Apple MacBook Pro (2017) with a 3.1 GHz Quad-Core Intel i7 and 16 GB 2133 MHz LPDDR3 memory. Since our calculations are doing in the Unity Editor and during the runtime of a simulation, the current delay is tolerable, but could be less acceptable for larger, more populated scenarios. Severson identified that this could be mitigated with larger spacing between nodes and aggregating entity representation to a higher unit level (2019) but this also would reduce the fidelity of the model. Regardless, looking for more efficient graph representations and scanning algorithms would be a beneficial area of study.

As Harder was able to automate the fire support planning for a deliberate attack, we feel that we could model the response of blue forces in a movement to contact operation. On contact with an enemy force, commanders have to decide whether to engage, bypass, or withdraw. We believe that this could be replicated for our automated blue forces and used in planning or improved wargaming. Inputs for this decision include the detected

enemy, estimate for the unseen threat, and environmental effects (terrain and weather), and commanders' intent. In our fog-of-war engagement (Chapter V Section E) we were able to provide a simulated version of all of those factors. While we weren't able to show engagement behavior, we noted that this could be added with route and objective maker reassignment. Implementing an automated movement to contact planner provides a direct application of our uncertainty threat graph and a logical improvement to WXXI.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. THE PREDICTOR EDITOR

This appendix outlines the input fields and buttons in the Predictor Editor for WXXI. To implement a Predictor, add the script Predictor2.cs to a GameObject in the WXXI seen. Figure 67 shows the top button panel for the Editor.

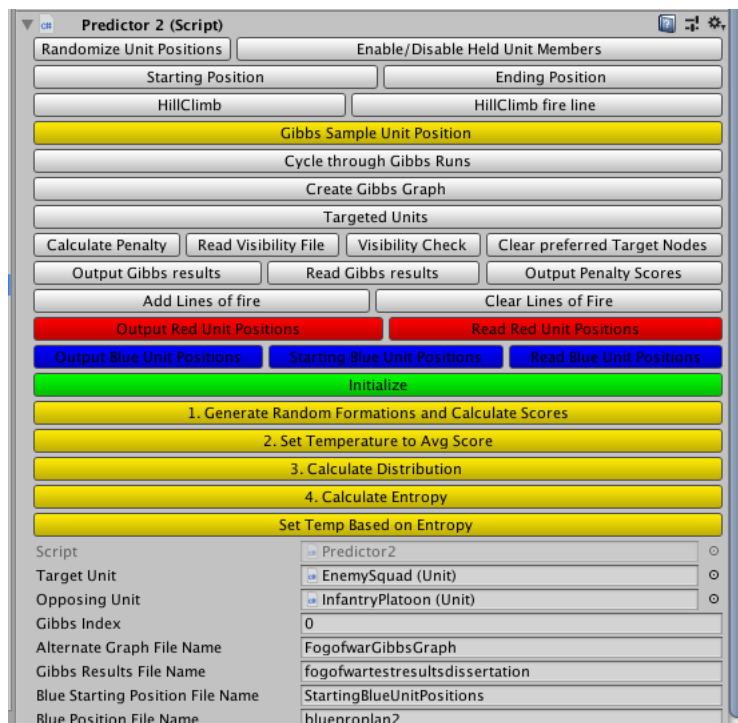


Figure 67. Predictor Editor Top.

Button and field descriptions for Figure 67 are as follows:

- Random Unit Positions: Assigns random positions to all entities in the unit designated in the Target Unit field.
- Enable/Disable Held Unit Members: When enabled prevents units in the Held Unit Members array from being assigned random positions or hill climbing.

- Starting Position: After hill climbing, moves Target Unit entities back to their original positions.
- Ending Position: After hill climbing, moves Target Unit entities to the positions that were the result of hill climbing.
- HillClimb: Makes all entities in the Target Unit perform hill climbing without the Line of Fire penalty.
- HillClimb fire line: Makes all entities in the Target Unit perform hill climbing with the Line of Fire penalty.
- Gibbs Sample Unit Position: After generating a candidate set of formations and calculating the distribution, selects a formation from the set according to its probability.
- Cycle through Gibbs Runs: After generating a candidate set of formations, iterates through each formation. Entity positions in the Scene window are updated for each formation. Index of the current displayed formation is shown in the console window and Gibbs Index field. It is possible to skip to a particular formation by changing the value in the Gibbs Index field to the preceding formation Index value and then hit this button.
- Create Gibbs Graph: Calculates node penalty values according to the candidate formation set VisibilityGraphs and their calculated probability in the Gibbs distribution. The result is saved to a text file designated by the Alternate Graph File Name field. A visualization of the new node values can be seen in the VisibilityGraphPre designated by the Gibbs Graph Index in the Astar Editor window. Note, output will not overwrite an existing text file. For best results, unique file names should be used, or the preexisting file should be deleted.

- Targeted Units: Determines Target Unit members that are targetable by the Opposing Unit. Results are displayed to the console window and appended to the “TargetedUnits.txt” file in the Resources folder.
- Calculate Penalty: Calculates the formation score of the entities in the Target Unit. Results are displayed in the Calculated Penalty Field.
- Read Visibility File: Used to read the precalculated visibility results into the visibilityArray [] for the VisibilityGraphPre graph indicated by the Astar Graph Index field. The visibility results are currently hard coded to the “VisibilityTest.txt” file in the resources folder, but this can be changed by editing the readVisibilityFile() function in the Predictor class.
- Visibility Check: Visualizes the threat on nodes by entities that are assigned to target them as either preferred targets or part of a FOF. Visualization is a sphere that increases as more threat is applied by assigned entities. Sphere color is assigned by the PREFERRED_TARGET_COLOR field.
- Clear preferred Target Nodes: Removes the Visibility Check visualization.
- Output Gibbs results: Writes the results of a Gibbs distribution calculation, formation scoring factors, and formation starting and ending positions to a text file designated by the Gibbs Results File Name field. Note, output will not overwrite an existing text file. For best results, unique file names should be used, or the preexisting file should be deleted.
- Read Gibbs results: Reads the results of a Gibbs distribution calculation, formation scoring factors, and formation starting and ending positions from a text file designated by the Gibbs Results File Name field.
- Output Penalty Scores: Outputs the formations scores from the candidate set to the text file “PenaltyScores.txt” in the Resources folder of the Unity project.

- Add Lines of fire: Adds visualization of entity lines of fire in the Unity Scene window.
- Clear Lines of Fire: Removes visualization of entity lines of fire in the Unity Scene window.
- Output Red Unit Positions: Appends the red entity positions to the text file “RedUnitPositions.txt” in the Resources folder of the Unity project. Note, output does not overwrite the existing text file. For best results, the preexisting file should be deleted.
- Read Red Unit Positions: Reads the red entity positions from the text file “RedUnitPositions.txt” in the Resources folder of the Unity project and repositions the red entities accordingly.
- Output Blue Unit Positions: Appends the blue entity positions to the text file “BlueUnitPositions.txt” in the Resources folder of the Unity project. Note, output does not overwrite the existing text file. For best results, the preexisting file should be deleted. The file can also be renamed and recalled with the Read Blue Unit Positions button. This button can be used during a simulation run to capture blue locations for later examination.
- Starting Blue Unit Positions: Reads the blue entity positions from the text file designated by the Blue Starting Position File Name field and repositions the blue entities accordingly. This is useful for returning the blue platoon to their original scenario starting positions.
- Read Blue Unit Positions: Reads the blue entity positions from the text file designated by the Blue Position File Name field and repositions the blue entities accordingly.
- Initialize: Starting point for building a Gibbs distribution. Reads the precalculated visibility file, assigns preferred targets and FOF to the red

entities, and ensures that Held Units is enabled. Must be pressed before moving on to the buttons for the Gibbs distribution

- 1. Generate Random Formations and Calculate Scores: Generates the number of candidate formations indicated by the Number Runs field. Formation Scores are calculated and displayed in the Penalty Scores Array.
- 2. Set Temperature to Avg Score: Calculates the Average formation score for the candidate set and sets the Temperature field to this value.
- 3. Calculate Distribution: Calculates the Gibbs Distribution using the Temperature field. Results are displayed in the Expo Penalty Divided by Sum Array.
- 4. Calculate Entropy: Calculates the entropy of the distribution. Results are displayed in the Entropy field.
- Set Temp Based on Entropy: Calculates the temperature for a desired entropy of the distribution. Desired entropy is entered in the Entropy field. The resulting temperature is displayed in the Temperature Field and the distribution probabilities are updated in the Expo Penalty Divided by Sum Array
- Target Unit: The defending Unit, fixed to red for our examples. Set by dragging a Unit GameObject from the Hierarchy window to the field.
- Opposing Unit: The attacking Unit, set to blue for our examples. Set by dragging a Unit GameObject from the Hierarchy window to the field.
- Gibbs Index: Current formation being displayed in the Scene window. Can be manually entered to skip to a particular formation in the candidate set (see Cycle through Gibbs Runs button above).

- Alternate Graph File Name: Designates text file name to store the calculated node values for the uncertainty threat graph. File saves to the project Resources folder. Note, output will not overwrite an existing text file. For best results, unique file names should be used, or the preexisting file should be deleted.
- Gibbs Results File Name: Designates text file name to store the results from the Create Gibbs Graph button. File saves to the project Resources folder. Note, output will not overwrite an existing text file. For best results, unique file names should be used, or the preexisting file should be deleted.
- Blue Starting Position File Name: Designates text file name to read the blue starting positions from. The file should be in the project Resources folder.
- Blue Position File Name: Designates text file name to read the blue positions from. The file should be in the project Resources folder.

The bottom of the Predictor Editor is shown in Figure 68.

Random Spawn Range	X	200	Y	0	Z	200
Random Spawn Off Set	X	0	Y	0	Z	0
Astar Graph Index	1					
Enemy Graph Index	2					
Gibbs Graph Index	3					
Calculated Penalty	2.708071e+10					
Number Runs	3					
▼ Penalty Scores						
Size	3					
Element 0	1.89831e+10					
Element 1	4.916884e+10					
Element 2	2.708071e+10					
▼ Starting Penalty Scores						
Size	3					
Element 0	-1.797167e+09					
Element 1	-3.296353e+10					
Element 2	-5.585061e+10					
File Read	☒					
Temperature	3.174422e+10					
Entropy	0.922018					
Highest Penalty Index	1					
▼ Expo Penalty Divided By Temp						
Size	3					
Element 0	1.818482					
Element 1	4.706324					
Element 2	2.34689					
▼ Expo Penalty Divided By Sum						
Size	3					
Element 0	0.2049757					
Element 1	0.5304875					
Element 2	0.2645368					
PREFERRED_TARGET_COLOR						
Targeted Penalty Weight	10000					
In Line Of Fire Penalty Weight	0					
Preferred Target Weight	1					
Field Of Fire Weight	100					
Distance From Commander Weight	1					
Dispersion Weight	1000					
Cohesion Weight	1e+07					
▼ Held Unit Members						
Size	2					
Element 0	RedSquadLeader v4 (1) (UnitMember) ○					
Element 1	OP RedStaticDefender v4 (1) (UnitMember) ○					
Hold Units	☐					
Max Distance From Commander	250					
Max Desired Distance	200					
Min Desired Distance	100					

Figure 68. Predictor Editor Bottom

Field descriptions for Figure 68 are as follows:

- Random Spawn Range: Designates Range for random repositioning for the Target Unit. This can also be set at the unit level. The Y value is left at zero.
- Random Spawn Off Set: Allows for an offset for the Random Spawn range from the Unit transform in the Scene Hierarchy. Best practice is to leave all values to zero.

- Astar Graph Index: Index of the VisibilityGraphPre for red threat towards blue in the Astar object.
- Enemy Graph Index: Index of the VisibilityGraphPre for blue threat towards red in the Astar object.
- Gibbs Graph Index: Index of the VisibilityGraphPre for displaying the calculated uncertainty threat graph in the Astar object.
- Calculated Penalty: Displays the calculated formation score.
- Number Runs: User input for desired number of formations in the candidate set.
- Penalty Scores: Array that displays the formation score for the final positions of each formation in the candidate set of formations.
- Starting Penalty Scores: Array that displays the formation score for the starting positions of each formation in the candidate set of formations
- File Read: Boolean value to indicate the precalculated visibility file has been read.
- Temperature: Displays temperature value.
- Entropy: Displays entropy value. It is also where the user inputs desired entropy for the distribution.
- Highest Penalty Index: The index of the highest scoring formation in the candidate set.
- Expo Penalty Divided by Temp: Array that displays the calculated Gibbs Score for each formation in the candidate set. It is only used for error checking.

- Expo Penalty Divided by Sum: Array that displays the calculated probability of each formation in the candidate set.
- PREFERRED_TARGET_COLOR: Sets the color for the Visibility Check visualization button.
- Targeted Penalty Weight: Scoring Formation function weight for cover penalty.
- In Line of Fire Penalty Weight: Scoring Formation function weight for in the line of fire penalty.
- Preferred Target Weight: Scoring Formation function weight for preferred Target nodes.
- Field of Fire Weight: Scoring Formation function weight for nodes in the FOF.
- Distance From Commander Weight: Scoring Formation function weight for distance from commander penalty.
- Dispersion Weight: Scoring Formation function weight for dispersion penalty.
- Cohesion Weight: Scoring Formation function weight for cohesion penalty.
- Held Unit Members: Array that holds the entities that are not part of the random placement and hill climbing process. It is used to set detected entities. Entity GameObjects can be dragged to open spots in the array from the Hierarchy window.
- Hold Units: Boolean value indicating hold units are enabled.
- Max Distance From Commander: The maximum desired distance from the commander in meters.

- Max Desired Distance: The maximum desired distance from other unit members in meters.
- Min Desired Distance: The minimum desired distance from the other unit members in meters.

APPENDIX B. THE PLAN PREDICTOR EDITOR

This appendix outlines the input fields and buttons in the PlanPredictor Editor for WXXI. To implement a PlanPredictor, add the script PlanPredictor.cs to a GameObject in the WXXI seen. Figure 69 shows the buttons and fields for the Editor.

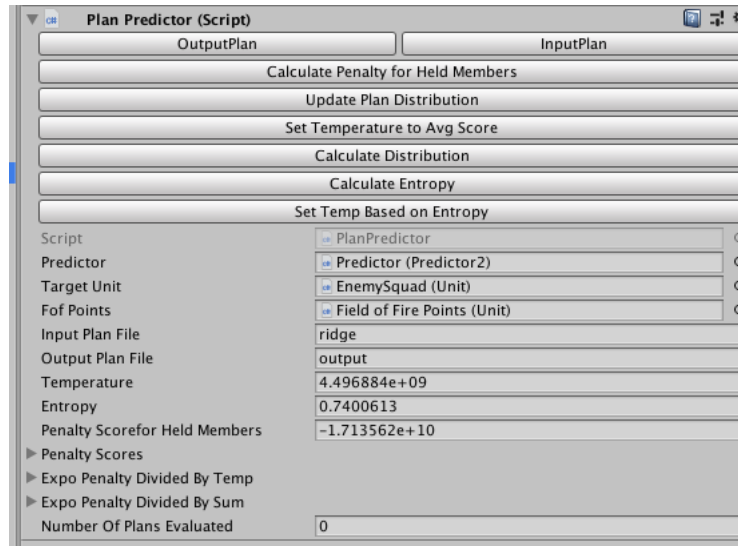


Figure 69. Plan Predictor Editor.

Button and field descriptions for Figure 69 are as follows:

- **Output Plan:** Writes the formation scoring factors to a text file designated by the Output Plan File field. Output include the scoring weights, ranges, and FOF point positions. Output will not overwrite an existing text file. For best results, unique file names should be used, or the preexisting file should be deleted.
- **Input Plan:** Reads in a formation scoring function from a text file designated by the Input Plan File field. Repositions the FOF points according to the input file. Once a plan is loaded, it must be initialized by the designated Predictor.

- Calculate Penalty for Held Members: Calculated the formation score for the held entities in the designated Predictor. This is used to simulate calculating the score for detected entities.
- Update Plan Distribution: Adds formation score for current plan to the formation score array and updates the probability distribution.
- Set Temperature to Average Score: Calculates the Average formation score the loaded plans and sets the Temperature field to this value.
- Calculate Distribution: Calculates the Gibbs Distribution using the Temperature field. Results are displayed in the Expo Penalty Divided by Sum Array.
- Calculate Entropy: Calculates the entropy of the distribution. Results are displayed in the Entropy field.
- Set Temp Based on Entropy: Calculates the temperature for a desired entropy of the distribution. Desired entropy is entered in the Entropy field. The resulting temperature is displayed in the Temperature Field and the distribution probabilities are updated in the Expo Penalty Divided by Sum Array
- Predictor: The designated Predictor for the scene. The Predictor is responsible for initializing each plan as it is loaded and identifying held entities.
- Target Unit: The defending Unit, fixed to red for our examples. Set by dragging a Unit GameObject from the Hierarchy window to the field.
- FOF Points: GameObject for the Hierarchical parent of the FOF points in the scene. We use a generic Unit object for our FOF points in the example.
- Input Plan File: Designates the text file name to read the formation scoring function factors from. The file should be in the project Resources folder.

- Output Plan File: Designates text file name to write the current scoring function to. File saves to the project Resources folder. Note, output will not overwrite an existing text file. For best results, unique file names should be used, or the preexisting file should be deleted.
- Temperature: Displays temperature value.
- Entropy: Displays entropy value. It is also where the user inputs desired entropy for the distribution.
- Penalty Score for Held Members: The calculated formation score for the detected (held) entities.
- Penalty Scores: An array holding the formation scores for each plan in the distribution.
- Expo Penalty Divided by Temp: Array that displays the calculated Gibbs Score for each formation scoring function in the distribution. It is only used for error checking.
- Expo Penalty Divided by Sum: Array that displays the calculated probability of each formation scoring function in the distribution.
- Number of Plans Evaluated: Total formation scoring functions in the distribution.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- AlphaStar Team. 2019. “AlphaStar: Mastering the Real-Time Strategy Game StarCraft II.” *DeepMind*. January 24, 2019. <https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii>.
- Ballanco, Edward. 2019. “We Need AN AI-Based Enemy Analysis Tool ... Now!” U.S. *Army War College War Room* (blog). January 16, 2019. <https://warroom.armywarcollege.edu/articles/enemy-analysis-tool-now/>.
- Bererton, Curt. 2004. “State Estimation for Game AI Using Particle Filters.” In *Proceedings of the AAAI Workshop on Challenges in Game AI*. <https://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-008.pdf>.
- Darken, Christian, and B Anderegg. 2008. “Particle Filters and Simulacra for More Realistic Opponent Tracking.” In *Game AI Programming Wisdom 4*, edited by Steve Rabin, 419-427. Boston: Charles River Media.
- Darken, Christian J., Daniel McCue, and Michael Guerrero. 2010. “Realistic Fireteam Movement in Urban Environments.” In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 9–14. AIIDE’10. Stanford, California, USA: AAAI Press. <http://dl.acm.org/citation.cfm?id=3014666.3014670>.
- Drain, Brendan. 2012. “League of Legends Tournament Cheaters Fined \$30,000.” *Engadget*. October 10, 2012. <https://www.engadget.com/2012-10-10-league-of-legends-tournament-cheaters-fined-30-000.html>.
- Glyde, Henry. 2010. “Lecture Notes On Statistical Mechanics.” University of Delaware. <http://www.physics.udel.edu/~glyde/PHYS813/Lectures/>.
- Granberg, Arron. 2016. The A* Pathfinding Project, version 3.8.2. Unity software asset. Accessed July 23, 2020. <http://arongranberg.com/astar/>.
- Groux, Christopher. 2018. “‘PUBG’ Radar Hack Ban Wave Sparks Esports Cheating Controversy.” *Newsweek*, December 22, 2018. <https://www.newsweek.com/pubg-radar-hack-ban-wave-esports-cheat-controversy-1269469>.
- Harder, Byron R. 2017. “Automated Battle Planning for Combat Models with Maneuver and Fire Support.” PhD diss., Naval Postgraduate School. <https://calhoun.nps.edu/handle/10945/52986>.

- Hladky, Stephen, and Vadim Bulitko. 2008. "An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games." In *2008 IEEE Symposium On Computational Intelligence and Games*, 39–46. Perth, Australia: IEEE. <https://doi.org/10.1109/CIG.2008.5035619>.
- Infinity Code. 2016. *Real World Terrain*. <http://infinity-code.com/assets/real-world-terrain> <http://infinity-code.com/assets/real-world-terrain> Unity software asset..
- Isla, Damian A, and Bruce M Blumberg. 2002. "Object Persistence for Synthetic Creatures." In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*. Bologna, Italy: Association for Computing Machinery.
- James, Alex, John Pierowicz, Mike Moskal, Timothy Hanratty, Dan Tuttle, Bob Sensenig, and Bill Hedges. 2017. *Assessing Consequential Scenarios in a Complex Operational Environment Using Agent-Based Simulation*. ARL-TR-9950. Aberdeen Proving Ground, MD: U.S. Army Research Laboratory. <https://apps.dtic.mil/sti/pdfs/AD1030073.pdf>
- Kott, Alexander, Rajdeep Singh, William McEneaney, and Wes Milks. 2011. "Hypothesis-Driven Information Fusion in Adversarial, Deceptive Environments." *Information Fusion* 12, no. 2 (April): 131-144. <https://www.sciencedirect.com/science/article/pii/S1566253510000771?via%3Dihub>.
- Lidén, Lars. 2002. "Strategic and Tactical Reasoning with Waypoints." In *AI Game Programming Wisdom*, edited by Steve Rabin, 211-220. Boston: Charles River Media.
- Maroon, Kenneth, and Christian Darken. 2019. "Predicting Enemy Squad Positions Based on Environmental Data, Unit Objectives, Tactics and Observed Entities." In *Proceedings of the International Conference on Social Computing Behavioral-Cultural Modeling, & Prediction and Behavior Representation in Modeling and Simulation*. Washington, DC. http://sbp-brims.org/2019/proceedings/papers/working_papers/Maroon_paper_17.pdf.
- Miller, David. 2016. "Hierarchical Task Network Prototyping in Unity3D." Master's thesis, Naval Postgraduate School. <https://calhoun.nps.edu/handle/10945/49346>.
- Pei, Annie. 2019. "This Esports Giant Draws In More Viewers Than the Super Bowl, and It's Expected to Get Even Bigger." *CNBC*. April 14, 2019. <https://www.cnbc.com/2019/04/14/league-of-legends-gets-more-viewers-than-super-bowlwhats-coming-next.html>.
- Severson, Peter. 2019. "Advanced Terrain Reasoning and Automated Maneuver Planning." Master's thesis, Naval Postgraduate School. <https://calhoun.nps.edu/handle/10945/62714>.

- Straatman, Remco, William van der Sterren, and Arjen Beij. 2005. "Killzone's AI: Dynamic Procedural Combat Tactics." In *Proceedings of the 2005 Game Developers Conference*. San Francisco: UBM Technologies.
- Thompson, Tommy. 2017. "The Perfect Organism: The AI of Alien Isolation." *Gamasutra*. October 31, 2017.
https://www.gamasutra.com/blogs/TommyThompson/20171031/308027/The_Perfect_Organism_The_AI_of_Alien_Isolation.php.
- Tozour, Paul. 2001. "Influence Mapping." In *Game Programming Gems 2*, edited by Mark DeLoura, 287-297. Boston: Charles River Media.
- . 2004. "Using a Spatial Database for Runtime Spatial Analysis." In *AI Game Programming Wisdom 2*, edited by Steve Rabin, 381-390. Boston: Charles River Media.
- U.S. Army. 2015. *Intelligence Preparation of the Battlefield/Battlespace*. ATP 2–01.3. Washington, DC: Headquarters Department of the Army.
https://armypubs.army.mil/epubs/DR_pubs/DR_a/pdf/web/atp2_01x3.pdf.
- U.S. Marine Corps. 2018. *Infantry Company Operations*. MCRP 3–10A.2. Washington, DC: Headquarters United States Marine Corps.
<https://www.marines.mil/News/Publications/MCPPEL/Electronic-Library-Display/Article/900521/mcrp-3-10a2-formerly-mcrp-3-10a1/>
- Weber, Ben G., Michael Mateas, and Arnav Jhala. 2011. "A Particle Model for State Estimation in Real-Time Strategy Games." In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Stanford, California. http://alumni.soe.ucsc.edu/~bweber/pubs/weber_aiide11.pdf.
- Zacny, Rob. 2014. "Bad to Worse: Allegations of Cheating at League of Legends Playoffs, Team SoloMid's Reginald Expresses View That Azubu Frost Abused the Stage Minimaps." *PCGamesN* (blog). May 29, 2014.
<https://www.pcgamesn.com/leagueoflegends/bad-worse-allegations-cheating-league-legends-playoffs-team-solomids-reginald-expresses-view-azubu-frost-abused-stage-minimap>.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California