APPLICATION OF THE MONTE-CARLO TREE SEARCH TO MULTI-ACTION TURN-BASED GAMES WITH HIDDEN INFORMATION

THESIS

Connor M. Pipan, Captain, USAF

AFIT-ENG-MS-21-M-072

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-072

APPLICATION OF THE MONTE-CARLO TREE SEARCH TO MULTI-ACTION TURN-BASED GAMES WITH HIDDEN INFORMATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Connor M. Pipan, B.S.E.E.

Captain, USAF

March 25, 2021

AFIT-ENG-MS-21-M-072

APPLICATION OF THE MONTE-CARLO TREE SEARCH TO MULTI-ACTION TURN-BASED GAMES WITH HIDDEN INFORMATION

THESIS

Connor M. Pipan, B.S.E.E.
Captain, USAF

Committee Membership:

Dr. Gilbert L. Peterson, Ph.D
Chair

Dr. Mark G. Reith, Ph.D
Member

Lt Col David W. King, Ph.D
Member

AFIT-ENG-MS-21-M-072

Abstract

Many military AI domains require planning actions for multiple units simultaneously in the presence of hidden information, similar to multi-action turn-based strategy games. The Monte-Carlo Tree Search (MCTS) algorithm has previously been applied to multi-action turn-based games, but comparatively little research exists applying it to multi-action turn-based games with hidden information. This thesis implements several Monte Carlo Tree Search (MCTS)-based agents in TUBSTAP, an open-source multi-action turn-based game, modified to include hidden information via fog-of-war. This thesis compares the performance of three hidden information MCTS approaches (Perfect Information Monte Carlo, Multi-Observer Information Set MCTS, and Belief State MCTS) and their suitability for multi-action turn-based games with hidden information. This comparison demonstrates that the Perfect Information Monte Carlo search outperforms the other algorithms significantly, likely due to domain complexity or implementation specifics.

Table of Contents

Page

Abstract iv

List of Figures vii

List of Tables viii

I. Introduction 1

1.1 Introduction 1
1.2 Problem Statement 2
1.3 Background to Research 3
1.4 Purpose of Study 3
1.5 Methodological Approach 4
1.6 Research Questions 4
1.7 Research Objectives 4
1.8 Results 5
1.9 Significance 5
1.10 Limitations 6
1.11 Summary 6
1.12 Document Overview 7

II. Background and Literature Review 8

2.1 Game Tree Search 8
2.2 Minimax Search 11
2.3 Monte Carlo Tree Search 13
2.3.1 Multi-Action Turn Based Extensions 15
2.3.2 Hidden Information Extensions 16
2.4 Summary 20

III. Methodology 21

3.1 TUBSTAP Modifications 21
3.2 Challenges 22
3.3 Perfect Information/Determinized MCTS 24
3.4 Multi-Observer IS-MCTS 25
3.5 Belief-State MCTS 26
3.6 Limitations 27
3.7 Summary 28

Page

IV. Results and Analysis 30

4.1 Experiment Design 30
4.2 Results 32

V. Conclusions 37

5.1 Future Work 37

Appendix A. Algorithms 40

1.1 Monte Carlo Tree Search 40
1.2 Negamax 40
1.3 MCTS UCT 40
1.4 Perfect Information/Determinized MCTS 42
1.5 MO-ISMCTS 44
1.6 BS-MCTS 46

Appendix B. Tables 48

Bibliography 52

List of Figures

Figure		Page
1	Starting TUBSTAP map (GPW_fujiki.tbsmap)	32

List of Tables

Table		Page
1	Unit Visibility Range	22
2	Historical Iteration Budgets	31
3	Experiment Results	32
4	Experiment List	50

APPLICATION OF THE MONTE-CARLO TREE SEARCH TO MULTI-ACTION TURN-BASED GAMES WITH HIDDEN INFORMATION

I. Introduction

1.1 Introduction

Wargames have been used for centuries by militaries around the world to teach and train battlefield tactics. Unlike traditional games, wargames are specifically described as multi-sided abstracted representations of armed conflict [1]. Unlike traditional games, wargames are designed partly to provide an accurate representation of combat, but also to encourage and incentivize strategic (or "down-board") thinking [1]. While modern wargames have diverged greatly through iterative rule changes and now come in many different forms, this evolution is apparent in the genre of turn-based strategy games.

Turn-based strategy games are games where players take turns to accomplish some strategic objective and are usually adversarial in nature. Chess is an example of such a game encouraging strategic "down-board" thinking [1], though modern turn-based strategy games are less abstract and usually feature many unique components of their own, such as individual unit hit points, map asymmetry, and/or specialized units [2].

Games have long been an area of interest to AI researchers. Skilled AI players often use a form of planning to identify effective actions, which requires an efficient search algorithm. However, traditional search approaches (e.g. DFS, A*) struggle with more complex domains[3], resulting in a need for a better search algorithm to improve performance. Multi-action turn based games, where players can take

multiple actions per turn, are particularly challenging due to the number of states and the number of subsets of actions that must be considered.

A common approach towards handling highly complex search domains is to use some sort of stochastic search algorithm. Stochastic search algorithms avoid searching the entire domain, ideally balancing exploration and exploitation to bias towards promising actions or states. However, certain factors (such the addition of hidden information) can greatly increase domain complexity, resulting in a combinatorial explosion that is far more difficult to search efficiently.

Additionally, in adversarial domains (where players compete against each other), usually some form of opponent model is needed to produce satisfactory results [3]. The search will be unable to return an optimal adversarial solution (the best sequence of actions leading to a goal state) if the algorithm that generated it assumes an opponent is playing poorly.

1.2 Problem Statement

Multi-action turn-based games are more complex than their single-action counterparts, since multiple subsets of actions must be considered when performing a search (versus considering single actions only). Domains with hidden information, such as games where certain information is hidden from the player or where actions can have uncertain outcomes, are likewise more complex than domains that are fully observable, since an algorithm must also consider all possible states the game could be in based on current and historical actions. Multi-action turn-based games with hidden information, as a result, are extremely complex and difficult to search effectively.

The Monte-Carlo Tree Search (MCTS) algorithm and its extensions have shown promise in highly complex domains. However, the relationship between MCTS search performance and the domain it is applied to is not well understood. Implementing,

testing and comparing the performance of several MCTS search modifications in a highly complex domain (such as multi-action turn-based games with hidden information) may improve understanding of the algorithm, its variations and may lead to the adoption or development of better search algorithms.

1.3 Background to Research

MCTS has been applied to many different domains, including multi-action turn-based games [2]. Though typically applied to deterministic, fully observable games, various extensions to the algorithm have been made for stochastic games and games with hidden information. However, the simplest variation, Determinized/Perfect Information Monte Carlo (PIMC), suffers from shallow search depth, strategy fusion and non-locality [4]. Many other extensions such as the Information-Set MCTS (IS-MCTS) [4], Belief-State MCTS [5], and others have been developed to address these shortcomings to various effect [4, 6].

1.4 Purpose of Study

There are many applications of strategy-game-playing artificial intelligences that are pertinent to the Department of Defense, such as command and control, and adversarial/defensive network operations. In real world applications, domain knowledge is rarely perfect. The real world is messy and random, which means practical AI agents need to be robust enough to handle situations where many of the variables involved are hidden, unknown or changing. A better understanding of hidden-information MCTS variants in multi-action turn based domain may lead to the development or adoption of better search algorithms that can search deeper and more efficiently, greatly increasing AI agent performance in a variety of applications.

1.5 Methodological Approach

This project involved the modification of TUBSTAP, an open-source fully-observable multi-action turn-based game, to feature hidden information via Fog of War. TUBSTAP is a turn-based strategy game where each player commands an army of units with the goal of eliminating the opponent's army. This project also implemented three game-playing agents based on the Perfect Information Monte Carlo, Multiple-Observer IS-MCTS, and Belief-State MCTS algorithms. Each agent played a series of matches against the others, with relative win rate taken as a performance metric. These matches also modified parameter values (specifically sampling and iteration count limits) to gauge their effect on the agents' performance.

1.6 Research Questions

The research questions this project intends to answer include:

- What are the relative strengths of the three algorithms tested (PIMC, IS-MCTS and BS-MCTS), and which performs best in multi-action turn-based domains with hidden information?
- What impact do determination, iteration and sample count settings have on these algorithms in this domain?

1.7 Research Objectives

This project includes several research objectives:

- Modify the TUBSTAP platform to feature hidden information via Fog of War.
- Develop a game-playing agent based on the Perfect Information Monte Carlo (PIMC) algorithm.

- Develop a game-playing agent based on the Multi-Observer Information-Set MCTS (MO-ISMCTS) algorithm.
- Develop a game-playing agent based on the Belief-State MCTS (BS-MCTS) algorithm.
- Perform simulations to compare the relative performance of the PIMC, MO-ISMCTS and BS-MCTS algorithms.

1.8 Results

In simulations, the PIMC-based agent significantly outperforms the others, winning 100% of all matches. The PIMC algorithm is known to outperform the IS-MCTS algorithm in sufficiently stochastic domains [7], and it is possible that multi-action turn-based games with hidden information are sufficiently complex that the supposed advantages of the IS-MCTS and BS-MCTS algorithms are not realized. However, a 100% win rate is unusual and has not been seen in previous research[7], which means this result should be viewed with skepticism, since the absolute dominance of the PIMC algorithm may indicate implementation complications giving unfair advantage to the PIMC algorithm or disadvantage to the IS-MCTS and BS-MCTS algorithms.

1.9 Significance

The extremely strong performance by the PIMC-based agent is anomalous and should be viewed with skepticism. With this in mind, it should be assumed that an accurate performance comparison between the tested algorithms was not obtained. However, two of the algorithms tested (MO-ISMCTS and BS-MCTS) are more complex and difficult to implement than the third (PIMC), and this project provides a practical consideration for future engineers and researchers of the relative speed and

implementation difficulty of these algorithms.

1.10 Limitations

The experiments for this project used a single, shared map layout and unit composition. Different, more varied test maps and unit compositions may be more representative of the domain and could affect experimental results, but these were ignored to reduce the number of variables that could impact performance, and to assist in finding a fair comparison between the algorithms.

Additionally, certain search factors for each agent (discussed further in Chapter 4) were left unmodified. Changing these values could impact search performance and experimental results- for example, improving performance by tuning the algorithm to strike a more favorable balance between exploration and exploitation.

1.11 Summary

This project attempted to compare the performance of several different MCTS-based game-playing agents in a multi-action turn-based game with hidden information. One of these agents (based on the PIMC algorithm) outperformed a MO-ISMCTS-based agent and a BS-MCTS-based agent in all matches played. These results strongly suggest an implementation error, although there is a possibility that other factors exist that favor the PIMC algorithm. Regardless, MO-ISMCTS and BS-MCTS are more complex algorithms and are more difficult to implement than PIMC, and the results of this project should be considered when implementing these algorithms in practical, real-world applications.

1.12 Document Overview

Chapter I provides an overview of the thesis as a whole, including the problem statement and overall research objectives.

Chapter II provides more background on the problem and research objectives, including AI applications in other games, other MCTS variations, and their applications.

Chapter III describes the methodology used for accomplishing research objectives, including pseudocode and high-level descriptions of the agents developed. Three different algorithms are compared in this chapter: perfect-information/determinized MCTS, multi-observer information set MCTS (MO-ISMCTS), and a Belief-State MCTS (BS-MCTS). It also describes experimental design and how results have been analyzed.

Chapter IV compares the performance of the algorithms tested and describes the results of the experiments, including detailed analysis of the results.

Chapter V provides a conclusion for the thesis, including a high-level overview of experimental results and scientific conclusions, as well as possible directions for future work or research.

II. Background and Literature Review

Multi-action turn based games are difficult for traditional AI search methods due to their extremely large search spaces. The addition of hidden information increases this complexity even further. Not only must an agent search over a large number of actions, but over a search space that includes all possibilities of what the hidden state may contain. Stochastic search methods such as the Monte Carlo Tree Search show promising performance in complex domains. Previous efforts have developed several extensions to MCTS to handle hidden information or multi-action turns, but there are few examples that combine both.

This chapter presents background on AI game applications, common game-tree search approaches, the Monte Carlo Tree Search, and variations of it applied in hidden information and multi-action domains.

2.1 Game Tree Search

Games have long been a staple of AI research as a way of measuring AI performance [3]. Goals in developing AI for games typically involve developing an agent that can outperform human beings, or an agent that can solve a game by finding the best action to take in any scenario [8].

In terms of AI for games, Allis [9] defined four pertinent domain characteristics:

- Perfect/Imperfect Information: Whether or not players have access to all information regarding the current game state. In a Perfect-Information domain, all variables are known to all players, and action results are strictly deterministic (and hence can be simulated by the players). Imperfect Information comes in two forms: hidden information (e.g. fog-of-war) or non-determinism (e.g. where the results of actions have a stochastic component, or only a chance of success,

such as a dice roll or card pull). Imperfect Information domains often result in larger search branching factors, since all possible outcomes from an action must also be explored.

Games of perfect information can generally be won using a "pure strategy", where a single best move can be chosen for any given game state. Imperfect information games, on the other hand, generally require "mixed" strategies, where moves are chosen according to some probability [9].

- Convergence: Whether the domain tends to converge to a small number of possible states or diverge into a large number. The state-space for any game can be imagined as a directed graph, where the nodes represent states and the edges represent actions that convert one state to another. Nodes can be separated into two sets that we'll call A and B: if there are more edges from A to B than vice-versa, then the domain is said to converge (else, it diverges or remains unchanged). A game can be both convergent and divergent depending on the stage.

  For example, chess is a domain exhibiting convergence. In chess, pawns must move forward or diagonally, which means they are incapable of returning to a previous position/state, reducing the number of legal states that can be achieved by moving them. Additionally, pieces that are captured are removed from the board and do not return- because there are fewer pieces, there are fewer possible states and thus the state space converges.

  Othello is an example of a domain exhibiting divergence: every move adds another piece to the board, which increases the number of legal moves (and states) that can be reached. However, Othello also exhibits convergence, transitioning near the end of the game when the board is nearly full (and thus leaving only a few possible moves).

For converging games, the number of terminating states is often a very small subset of states in the state-space, which allows for the creation of endgame databases that can improve search performance through the use of what is essentially a lookup table [10]. Due to the number of potential states, this approach generally infeasible in diverging or unchangeable games [9].

- Sudden Death: Whether the game is quickly capable of reaching a terminating state given the creation of a certain pattern. For example, the game Go-moku ends when all spaces are occupied without any player winning, or when one creates a line of five stones in their team's color (which can happen very early, hence "sudden death"). Games that do not exhibit this quality are known as "fixed-termination" games and tend to last approximately the same number of turns regardless of the strategy employed.

  The presence of the Sudden Death property can be greatly beneficial for improving performance. These strategies usually lie relatively shallow in the search tree- approaches such as a Killer Move heuristic can bias search towards highly advantageous moves that result in decisive victories [9].

- Complexity: Complexity of a game domain is represented in two ways: state-space and game-tree complexity. State-space complexity is number of possible legal game positions reachable from some initial position. This is useful for providing a bound for complexity when attempting to solve a game through enumeration. In complex domains, this number can be difficult to compute and is usually approximated.

  Game-tree complexity is the number of nodes in the solution search tree from the initial position of the game (also usually approximated). Since different actions may lead to the same state along different paths, this value is usually

larger than the state-space complexity. This is a better representation of the size and difficulty of the search and, in the case of Minimax search, is a reasonable estimate of the size/depth of the Minimax search tree.

There are several methods for managing complex domains. One approach is to simply avoid the use of a prominent search component and instead use a rules-based agent, picking the best action according to a set of rules and previously-defined expert knowledge given a known state. One example is an agent developed by Cutright [11], where the agent is series of largely individual modules responsible for separate decision points. Another approach is to use learning or evolutionary algorithms, such as in the case of Hearthstone as implemented by Lee, et al [12]. Another is to use a stochastic search algorithm like the Monte-Carlo Tree Search (MCTS) to avoid searching a significant portion of the domain- this approach will be discussed more in-depth later in the chapter.

2.2 Minimax Search

For game-tree search, single agent AI searches encode the current environment state as a root node in a graph. From this, edges are generated for legal moves or actions that result in additional states. Iterative application of this process results in a tree that can be explored by common search algorithms e.g. DFS, BFS, or Best-FS. Accordingly, each layer in the search tree represents possible states the environment can be in some number of actions ahead of the current state (referred to as the search depth). The number of possible actions at each node is called the *branching factor*.

One optimal search strategy for deterministic games is Minimax search [3] . Minimax is a recursive search that assigns a utility value for each node in the tree, backpropagated from some terminal state. It offers rudimentary opponent modelling by selecting the action with the optimal value for the acting player, minimizing utility

value on an "opponent's" turn while maximizing utility on the player's turn (hence the name). This is commonly implemented through Negamax, a simplified Minimax variant.

Algorithm 1 Negamax

```
1: function NEGAMAX(node, depth, color)                                    ▷
2:     if depth = 0 or node is terminal then
3:         return Color* Heuristic value of node
4:     end if
5:     value := −∞
6:     for each child of node do
7:         value := max(value, −negamax(child, depth − 1, −color))
8:     end for
9:     return value
10:
```

The drawback of Minimax is that it performs a complete depth-first search of the game tree [3] , resulting in a high time and space complexity. A common Minimax improvement used to reduce the size of the search tree is Alpha-Beta Minimax [19]. Alpha-Beta Minimax improves Minimax by reducing the number of nodes evaluated by the search tree. It does so by halting evaluation in a sub-tree when there is a guaranteed worse outcome than a previously examined move. With best-case move ordering, only $O(b^{m/2})$ nodes (where b is the average branching factor and m is the maximum tree depth) need to be searched to identify the best move, versus $O(b^m)$ with standard minimax. This reduces the branching factor from b to $\sqrt{b}$, allowing alpha-beta minimax to search a tree twice as deep as standard minimax in the same amount of time [3].

Minimax with Alpha-Beta has been applied to TUBSTAP[13], though the domain remains complex enough that additional move ordering heuristics [13] were needed to adequately reduce the search space. This involved three types of pruning: fixing the order in which units were allowed to move, applying selective action generations, and limiting the number of moving units in each search. As Sato, et al. [13] state, forward

pruning heuristics such as these carry some amount of risk due to the possibility of overlooking important or critical moves- however, decreasing the number of edges in the game tree allows the algorithm to search deeper. This improves the algorithm's look-ahead capability, allowing an agent to consider more future moves taken by its opponent, improving performance.

Minimax has also been applied to stochastic domains through Expectiminimax [3] . This augments the Minimax algorithm through the addition of "chance" nodes, which represent random events or stochastic outcomes (e.g. the roll of a die). These nodes are weighted with the sum of values over all outcomes. This improves performance in hidden information domains (since the algorithm is now capable of taking random actions or outcomes into account) at the cost of increased memory complexity (since additional nodes must be generated for chance events).

2.3 Monte Carlo Tree Search

Stochastic search methods such as the Monte Carlo Tree Search (MCTS) have been shown to be effective in game domains [14]. MCTS is a stochastic search algorithm that builds a search tree by Monte-Carlo sampling from a distribution of actions and gradually biasing towards actions that offer the most promising result. The first broad success of MCTS in games was for the game of Go [15]. The algorithm has four basic steps:

1. **Selection**, where (starting at the root node), successive child nodes are selected until a leaf node is reached, biasing in favor of promising nodes.

2. **Expansion**, where additional child nodes are created from the leaf node depending on available actions (and provided the leaf node does not represent a terminating state).

3. **Simulation**, where a game is simulated starting from the leaf node until it reaches some terminating state, usually choosing random (but valid) actions.

4. **Backpropagation**, where the results of the simulation are used to update the values of the nodes between the child and the root.

MCTS has several benefits that make it appealing for complex domains:

- It is an anytime algorithm (which means the search can be halted at any point and still return the best action found up until that point).
- More computing power generally leads to improved performance.
- It requires little domain knowledge past a basic understanding of state and valid actions [14], although domain-specific heuristics can improve performance further.
- Over enough iterations it converges towards an optimal Minimax solution while requiring far less memory.

Algorithm 2 General MCTS Approach

```
1: function MctsSearch(s_0)
2:     create root node v_0 with state s_0
3:     while Within computational budget do
4:         v_l ← TreePolicy(v_0)
5:         Δ ← DefaultPolicy(s(v_l))
6:         Backup(v_l, Δ)
7:     end while
8:     return a(BestChild(v_0))
9:
```

Detailed pseudocode can be viewed at appendix 1.3. *TreePolicy* (i.e move selection) is an important factor in search performance. MCTS implementations commonly use Upper Confidence Bounds for Trees (UCT) [15]. An upper confidence

bound score is computed for each child node, and the node with the highest (maximizing) score is selected for expansion. This approach helps balance exploration and exploitation. *DefaultPolicy* (i.e. expansion) is how child node are selected from the leaf- children are usually selected uniformly at random, although more intelligent policies can be chosen instead. *Backup* backpropagates utility values from the children to the parents.

2.3.1 Multi-Action Turn Based Extensions

MCTS has been applied to TUBSTAP, though only as a perfect information domain [2]. Fujiki, et al. [2] implemented several MCTS variants. One interesting result was that their depth-limited Monte Carlo agent with a combined Depth-Limited Monte Carlo Method and Attack Action Search heuristic (DLMS+AAS) outperformed unenhanced, UCT MCTS agents. DLMS is implemented in simulation, limiting the number of simulations by implementing a depth cut-off and returning a board state evaluation value. AAS is implemented in the expansion step, limiting the length of the action list (and number of child nodes). This suggests that even the perfect information domain remains complex enough that tree pruning and move ordering can significantly improve performance.

MCTS has also been applied by to Hero Academy[16], another adversarial turn-based game with high complexity. Justesen et al tested several MCTS variants as well as an evolutionary algorithm they called Online Evolutionary Planning. Similarly to Fujiki, et al.[2], they found that the constrained MCTS variations tested significantly outperformed the unmodified MCTS, reinforcing the suggestion that an informed, guided or otherwise limited search further improves MCTS performance in complex domains.

2.3.2 Hidden Information Extensions

A common approach to stochastic, hidden information games is to "determinize" them, translating the game into a deterministic domain so a traditional MCTS or Minimax search can be run. This approach runs many searches over a number of separate determinizations, eventually allowing the algorithm to choose the best apparent move. Determinization works well if the hidden information does not influence the game until it is revealed- however, it can be ineffective in certain domains (such as card games, where hidden cards are most often present in the deck or an opponent's hand), leading to strategy fusion and non-locality.

Strategy Fusion occurs when the search selects different actions depending on the determinization, even if the states associated with those actions are indistinguishable from the player's point of view- for example, playing a different card depending on the cards in an opponent's hand, despite those cards not being visible. Non-Locality occurs when unlikely determinizations have an outsized effect on the search process- for example, performing a search where one of the underlying assumptions is that the opponent possesses a game-winning card, but refuses to play it. As found by Fernandes in the study of several card games [6], certain domain factors such as high leaf correlation (where outcome does not easily change late in the game) and high disambiguation factors (where hidden information is gradually revealed over the course of the game) can minimize these effects, but not eliminate them.

When applied to MCTS, this is known as a Perfect Information Monte-Carlo (PIMC) or Determinizing MCTS. Cowling and Prowley [17] investigated the use of PIMC for Magic: The Gathering, believing it to a good candidate for PIMC methods due to high leaf correlation. These authors tested several MCTS variants and found that enhanced variants outperformed basic MCTS agents but found that no individual enhancement significantly outperformed the others, and their combination did not

further improve playing ability to a significant degree. This represents a pattern expressed by Browne [14] and demonstrated by Fernandes [6]: the relationship between the performance of MCTS and the domain its applied to is still not well understood, and the effectiveness of a heuristic in one domain does not currently guarantee its effectiveness in another.

As mentioned, one of the primary drawbacks of a PIMC is its susceptibility to strategy fusion and nonlocality. The Information-Set Monte Carlo Tree Search (IS-MCTS) [4] attempts to address the issue of strategy fusion by searching over information sets (a collection of all possible states) instead of single states. IS-MCTS also searches over a single tree (instead of a collection of trees relating to specific determinizations), which allows it to search deeper, improving performance. In the domain of Mini Dou Di Zhu, IS-MCTS was able to significantly outperform PIMC [4].

This "single-observer" IS-MCTS (SO-ISMCTS) [7], notionally provides an advantage over PIMC and other variations by searching a single tree deeply instead of multiple trees shallowly. Instead of branching to every legal move, the availability of a branch depends on the current determinization (hence "single observer"). For PIMC, it is important to select a balanced number of determinization and MCTS iterations for single game tree- as long as these values are sufficiently large, their precise value does not have a significant effect on play strength. However, in the domain of "Love Letter" [7], SO-ISMCTS did not show any significant improvement over PIMC- in fact, at high iteration values PIMC outperformed SO-ISMCTS, with SO-ISMCTS barely outperforming simple knowledge-based agent. Omarov, et al. [7] suggest that in a sufficiently stochastic domain such as "Love Letter", SO-ISMCTS does not offer a significant improvement over other MCTS approaches or simpler agents (although SO-ISMCTS may be more effective in more deterministic domains).

IS-MCTS has been applied to other complex games of hidden information. One example is Pokémon, by Ihara, et al. [18]. Unlike most other discussed MCTS applications, Pokémon represents a particularly complex domain due to the high number of random chance nodes present in the search. Again, IS-MCTS agent was able to outperform PIMC, winning slightly more than half of the games played. Since Pokémon is a complex game with many possible determinizations, the authors similarly suggest this may be due to IS-MCTS using computational resources more effectively (as well as being less susceptible to strategy fusion).

IS-MCTS still suffers from nonlocality issues, which can only be addressed by inference and opponent modelling. One approach is to combine MCTS with Minimax searches, as investigated by Baier and Winands [19]. Minimax searches represent a basic form of opponent modelling- the "current" player aims to take the best ("max") action, after which a simulated opponent takes the least effective ("min") action (which can be considered the most effective action from the opponent's perspective). The Minimax search can be implemented in several places, such as during the rollout phase (MCTS-Informed Rollout), in place of the rollout phase to terminate rollouts early (MCTS-Informed Cutoff), or prior to bias move selection to bias towards more favorable modes (MCTS-Informed Priors). However, these variations can have difficulty in domains with large search spaces (such as those involving hidden information) or when Minimax searches are also used for state evaluations due to the high cost of repeated Minimax searches.

One IS-MCTS approach is the Re-Determinizing IS-MCTS (RIS-MCTS), as introduced by Goodman [20]. By default, IS-MCTS suffers from information leakage from the root player, "forcing" opponents to play certain moves regardless of the actions of the root player, suggesting different actions for different opponents when using the same information set, regardless of whether that information is actually present to

the opponent. This can result in strategy fusion. RIS-MCTS attempts to address this by re-determinizing hidden information from the perspective of the current player at each node of the search (hence the name).

IS-MCTS can also be combined with a PI-MCTS to form a Semi-Determinized-MCTS (SDMCTS), as introduced by Bitan and Krays [14[21]. IS-MCTS by default assumes that every game state in the current information set has the same probability of being the current game state, which is not always the case. By using an opponent's behavior to predict future moves, the number of states in the information set can be reduced. SDMCTS generates a predictive model of an opponent's actions using historical behavioral data- it then uses these predictions to build a determinization on which simulations can be run. These predictions help reduce the uncertainty an agent is dealing with, reducing the search space and improving performance. To do this, SDMCTS searches for an optimal strategy given an opponent's previous actions, and then estimates the reward for each possible action the opponent has. MCTS simulations are then run using only this strategy/information and the most optimal actions.

The Belief-State Monte-Carlo Tree Search (BS-MCTS) is another approach proposed by Wang, et al. [5]. Like IS-MCTS, BS-MCTS builds a single search tree using a collection of states; however, BS-MCTS attempts to address some of the shortcomings of IS-MCTS by pairing these states with probabilities. Unlike PIMC or IS-MCTS, which typically use a tree policy like UCT, the BS-MCTS search is instead guided by these probabilities, biasing towards states that appear to be more likely. The algorithm uses online learning to update these probabilities during play, improving performance.

2.4 Summary

This chapter discussed game-tree search, including challenges, different approaches and related work for handling game-tree search in complex domains. The addition of hidden information and/or stochasticity can greatly increase the complexity of a domain. Multi-action turn-based strategy games are particularly complex. However, stochastic search algorithms such as the Monte Carlo Tree Search offer many possible benefits over traditional search algorithms in these complex domains.

III. Methodology

Multi-action turn-based strategy games represent a complex domain that traditional, deterministic search algorithms struggle with. The Monte Carlo Tree Search (MCTS) is a stochastic search algorithm that has proven beneficial in complex domains (including multi-action turn-based strategy games), and MCTS extensions for domains with hidden information have also proven effective in other games. However, little research exists to evaluate the performance of hidden-information MCTS extensions in domains that feature both multi-action turns and hidden information.

This section details the modifications made to the Turn-Based Strategy Academic Platform (TUBSTAP) and the implementation details to address both multi-action turns and hidden information. These agents extend one of three algorithms: Perfect-Information/Determinized MCTS (PIMC), an Information-Set MCTS (IS-MCTS) variation known as the Multi-Observer IS-MCTS (MO-ISMCTS), as well as the Belief-State MCTS (BS-MCTS).

3.1 TUBSTAP Modifications

The Turn-Based Strategy Academic Platform (TUBSTAP) is an open-source multi-action turn-based strategy game. Two opposing players command a small army (approximately 5-10 units) with the goal of eliminating the army of the other player. Armies are composed of a mix of units, each with different abilities and strengths/weaknesses to units of different types, which are compounded by the type of terrain the unit occupies. TUBSTAP is normally a deterministic, fully-observable domain, but this project implements hidden information through the use of Fog of War. With Fog Of War, each unit has a visibility range that dictates how much of the map it can see. Enemy units within range are visible to the player, while enemy units outside

of it are not. For this implementation, unit visibility is strategic: enemy units are considered visible to all friendly units if they are in the line of sight for any friendly unit, and any friendly unit can act on any visible enemy unit (provided the enemy unit is in attack range).

The visibility range of each unit depends on its type. For this project, the chosen visibility ranges are intended to abstractly represent real-world capabilities, given the unit's operating domain and sensor capabilities. For example, infantry units have a very short (1-square radius) visibility range, while fighter jets have a very large one (5 squares).

Table 1: Unit Visibility Range

Unit Type	Visibility Range (tiles)
Fighter Jet	5
Attack Jet	5
Tank	2
Artillery	2
Anti-Air	2
Infantry	1

3.2 Challenges

While the MCTS and variants are generally domain-agnostic, several challenges arise when attempting to implement it within a hidden-information multi-action turn-based domain. Fujiki, et al. [2] have previously described many design components in turn-based strategy games that are worth considering, but the most significant for this application is move ordering. In TUBSTAP, units cannot occupy the same spaces as other units, and outcome of any engagement is dependent on the units involved (for example, anti-air units are more effective against flying units than armor). In a naïve implementation, one may attempt to enumerate all permutations of action subsets, with each subset leading to another state in the same way a single action

leads to a new state in a single-action domain. However, the branching factor renders this approach infeasible. For example, with an army size of 4, a board size of 8x8 and limiting limited units to only movement actions (assuming each unit can only traverse about half of the board), there are approximately $(4 * 29)! = 3.393 * 10^190$ permutations that must be considered. In comparison, Chess has a branching factor of about 35, while Go has a branching factor of about 200 [22].

A better approach, used by this implementation and the perfect-information MCTS implementation developed by Fujiki et al [2], is to instead treat the multi-action turn-based domain as a single-action turn-based domain where players can take many turns in a row. Here, instead of considering permutations of actions, only the single actions available to each unit are considered and used as branches in the tree. Using the previous example (a 4-unit army that can move approximately over half of an 8x8 board), the branching factor for each node is reduced to 32-3=29. In practice, for an 8x8 board with an army size of 8, the branching factor is closer to 200- greater, but still far more manageable.

Additionally, it is far easier to update the underlying determinization and information sets using the actions of single units only, and still allows the algorithm to take advantage of different move orderings and emergently superior strategies (for example, allowing a scouting unit to move first to reveal an enemy unit, which can then be attacked by an artillery unit). While there are still a great number of permutations, not all are visited, and an appropriate reward function allows the algorithm to balance exploration and exploitation. This approach is straightforward to implement and requires little modification to the underlying algorithms. In the case of the MO-ISMCTS, the biggest difference is that this implementation descends several nodes before switching decision trees, instead of alternating every turn. The BS-MCTS functions as expected (albeit selecting several rounds of opponent guessing or

opponent predicting in a row).

However, updating the information sets between the trees and the underlying determinization appropriately remains difficult. In the hidden-information domain each unit has a different view distance for revealing fog-of-war, which means unit placement also affects which enemy units can be seen and attacked. The order in which actions are executed can drastically impact the underlying information set by revealing enemy units or revealing a player's own units to the enemy.

3.3 Perfect Information/Determinized MCTS

One of the simplest means of adapting the MCTS to hidden-information domains is the Perfect Information/Determinzed MCTS (PIMC) [4]. PIMC works by running several separate MCTS searches on different determinizations of the underlying state. In effect, PIMC makes an assumption for hidden states and the results of actions with stochastic outcomes, then runs a traditional MCTS search as if the domain were fully observable. PIMC will use multiple determinizations, effectively running many shallow MCTS searches in sequence. Over enough determinizations and iterations, PIMC may converge to an optimal action. However, it is susceptible to strategy fusion and nonlocality, which can reduce the quality of the search.

Strategy Fusion occurs when different actions are chosen depending on the determinization, even if the states should be indistinguishable from the player's point of view. Non-locality occurs when unlikely determinizations have an outsized effect on the search process, despite a player's ability to direct play to or away from the state (such as an opponent refusing to make a game-winning move).

The PIMC implementation used here is a modified version of the perfect-information Monte Carlo previously implemented by Fujiki, et al. [2]. One of the primary benefits of the PIMC is its ease of implementation. This implementation creates a random

determinization, then runs a MCTS search on that determinization using the MCTS algorithm included in the TUBSTAP source, allowing it to be run in the hidden information domain.

Detailed pseudocode can be viewed at appendix 1.4.

3.4 Multi-Observer IS-MCTS

IS-MCTS attempts to address the issue of strategy fusion by searching a single tree of information sets deeply, rather than many trees of game states shallowly. An information set is a collection of all possible states the game could be in given the actions leading to the node in the search tree. For each iteration, IS-MCTS chooses a determinization at random, using only nodes and actions compatible with said determinization. Each action from the root creates a new node with an information set representing all possible states resulting from that action. Unlike PIMC, this approach uses a single search tree changing the determinization each iteration (selecting only actions compatible with that determination) instead of multiple search trees each running a specific determinization with multiple iterations. In theory, this allows IS-MCTS to search deeper than PIMC, potentially returning a better search result.

However, IS-MCTS still suffers from nonlocality issues, which can only be addressed through inference and/or opponent modelling. A variation called the Multi-Observer ISMCTS (MO-ISMCTS) was introduced by Whitehouse alongside SO-ISMCTS [4]. MO-ISMCTS attempts to address strategy fusion and non-locality by building a separate search tree for each player. Each search tree is descended simultaneously, and the information sets are made from the perspective of the acting player. This improves performance by eliminating strategy fusion (since partially observable actions lead to a unique node in the search tree), while also offering rudimentary opponent modelling (potentially reducing nonlocality) [4]. MO-ISMCTS consumes more mem-

ory than IS-MCTS, but can outperform it given enough computation time [4]. The implementation used here was written specifically for this project and is based on the pseudocode provided by Whitehouse [4].

Detailed pseudocode can be viewed at appendix 1.5.

3.5 Belief-State MCTS

The Belief-State Monte-Carlo Tree Search (BS-MCTS) is another approach proposed by Wang, et al. [5] designed to address some of the shortcomings of the IS-MCTS. BS-MCTS works by creating a belief-state tree. Like IS-MCTS, BS-MCTS uses multiple states per search node. However, unlike IS-MCTS (where each node contains an information set containing possible states), each node in the BS-MCTS instead contains a set of beliefs (known as a belief state). Each belief is a tuple containing a state and the probability the state represents the underlying reality- in other words, the probability that a particular belief is true [5].

Instead of using UCT, the search is guided by these beliefs, ideally biasing towards states with a higher probability of being true. BS-MCTS starts with an additional step - sampling- which initializes the search with a variety of possible states, which can be generated randomly or through some sort of inference. Each belief is initially given an equal probability, and then all are updated incrementally through search using various heuristic methods. For an adversarial domain like TUBSTAP, this comes in two forms: opponent guessing and opponent predicting. The method used depends on the position in the search tree and whether it is the player's turn or the opponent's. Opponent Guessing is used for states belonging to the player. It is an online learning algorithm that attempts to estimate the beliefs in terms of their rewards. Opponent Predicting is used for states belonging to the opponent. Opponent Predicting works by calculating probabilities depending on the utility of the states in

the node, assuming that an opponent will tend to pick favorable moves but will not always pick a move that is optimal [5].

Like MO-ISMCTS, BS-MCTS attempts to solve the issues of Non-Locality and Strategy Fusion. According to the authors, Strategy Fusion is largely the result of the search overestimating node value because the determinized search assumes different strategies can be chosen in different situations. BS-MCTS addresses this by initially assuming both nodes are equally likely, then assigning correct values after sufficient sampling has been performed. However, like IS-MCTS, BS-MCTS does not perfectly address the issue of non-locality except in cases where a perfect inference model can be developed, though better models can reduce the impact of the problem [5]. Even so, the BS-MCTS has been shown to significantly outperform MO-ISMCTS in the domain of Phantom Go [5].

The implementation used here was written specifically for this project and is based on the pseudocode provided by Wang, et al. [5]. Detailed pseudocode can be viewed at appendix 1.6.

3.6 Limitations

There are several factors which may impact the quality of the search. The primary advantage of the IS-MCTS over PIMC is its ability to search deeper in the tree than a standard PIMC search, given the same number of computation cycles. With multi-action turn-based games, search depth is important- a greater search depth allows for more permutations of unit actions allowing for the discovery of more strategies. However, when compared to a true single-action turn-based domain, multi-action domains will need to search far deeper into the tree to converge to a minimax optimum. This reduced look-ahead may make an these algorithms susceptible to traps.

Additionally, while MO-ISMCTS will search deeper given the same number of

computation cycles, the quality of the search is still dependent on the determinization. If the determinizations is inaccurate, the quality of the search is likely to be degraded. Depending on the environment certain assumptions can be made- for example, in a competition it may be fair to assume the opponent will start in reversed but identical position from our own with the same force makeup- but these assumptions are not likely to hold up across multiple environments; after all, fair fights are rare in the "real world".

Furthermore, MO-ISMCTS is not guaranteed to outperform PIMC if the domains are sufficiently stochastic [4]. TUBSTAP is a deterministic domain though it does possess limited, optional stochastic elements optionally applied at game start. However, even without these the domain remains complex. Even a somewhat informed determinization may be worth little more than a guess compared to the space of possibilities, which may close the gap between PIMC and IS-MCTS performance.

3.7 Summary

This chapter discussed implementation details, including modifications to the TUBSTAP platform and descriptions of the hidden information MCTS extensions used. TUBSTAP is a multi-action turn-based strategy game that is notionally fully observable. However, the platform was modified for this project to feature hidden information via Fog of War, which prevents players from viewing enemy units outside a certain visual range.

The MCTS extensions used are the Determinized/Perfect Information Monte Carlo (PIMC), Multi-Observer Information Set MCTS (MO-ISMCTS), and the Belief-State MCTS (BS-MCTS). PIMC is a simpler algorithm, but is incapable of searching deeply in the search tree and suffers from Strategy Fusion and non-locality. MO-ISMCTS searches deeper and addresses Strategy Fusion and Non-Locality, but is

slower and more complex. BS-MCTS also searches deeper and uses beliefs generated during play to guide the search. It addresses the issue of Strategy Fusion, but does not fully address the issue of non-locality.

IV. Results and Analysis

This section discusses experimental results. Using TUBSTAP's Autobattle system, each agent was compared against the other in 100 simulated matches. Experiments were run using iteration limits of 1000, 5000, 10000 and 20000, kept equal between each agent (accounting for differences between the algorithms). The relative win rate was used as a performance metric.

In these experiments, the PIMC agent defeated both the MO-ISMCTS and BS-MCTS agents in all matches, resulting in a 100% win rate. PIMC is known to outperform IS-MCTS in some domains [7], but the dominance of PIMC in these experiments may indicate an unknown issue with the implementation of the algorithms in this domain.

4.1 Experiment Design

TUBSTAP features a robust auto-battle system to compare agent performance, allowing AI agents to play repeated matches against the others, with the results output to an external log file. Each battle has specified starting state. This state is loaded from a file and consists of two primary components: the map and the units. The map represents the game board and consists of a two dimensional grid of squares. Each square is assigned one of several terrain types (such as field, forest or road), which influence unit move distances and attack advantage. The units are the pieces available to either player. Each unit has an assigned type (such as infantry, armor or armor), each with different movement ranges and vision ranges and attack power. An army is the set of units available to a certain player, mutually exclusive. These experiments utilized the default TUBSTAP autobattle settings, utilizing a single map and unit set. While the autobattle system allows initial unit positions and unit HP

to be shuffled per game, these options were not used. This design was chosen to minimize the number of variables that could impact playing performance in order to better compare the performance of each algorithm- more is discussed in Chapter V.

100 simulated matches were run between each pair of agents, with agents switching sides after 50 rounds. Each match was limited to a maximum 50 turns. No time limits were implemented. For each experiment, each agent shared the same iteration limit as its opponent.

Additional experiments were run between the PIMC and BS-MCTS, and each agent against itself to ascertain the impact of sampling/determinization count with performance. For these experiments, each agent was limited to 100 iterations with 10, 50, and 100 samples/determinizations allowed, for a total of 1000, 5000, 10000 and 20000 iterations. Although there is no fixed standard, these values were chosen to approximately match iteration budgets seen in previous research. However, values higher than 20000 were ignored for these experiments, since higher iteration budgets equate to linear increases in computation time as well as diminishing returns in performance [6].

A full list of experiments can be found in Appendix B. In all experiments, relative win rate was used as a performance metric. These experiments were performed on a computer with a AMD 3950X 16-Core/32-thread processor (approximately 4 GHz at sustained boost clocks) with 32 GB DDR4 RAM.

Table 2: Historical Iteration Budgets

Paper	Iteration Budget
Whitehouse [4]	2500, 5000, 10000, 15000
Fernandes [6]	500, 2500, 10000, 20000, 30000, 40000, 50000
Wang, et al. [5]	100,200,400,800,1000,2000,2500,5000,20000,40000,80000,100000,200000
Ihara, et al. [23]	1000

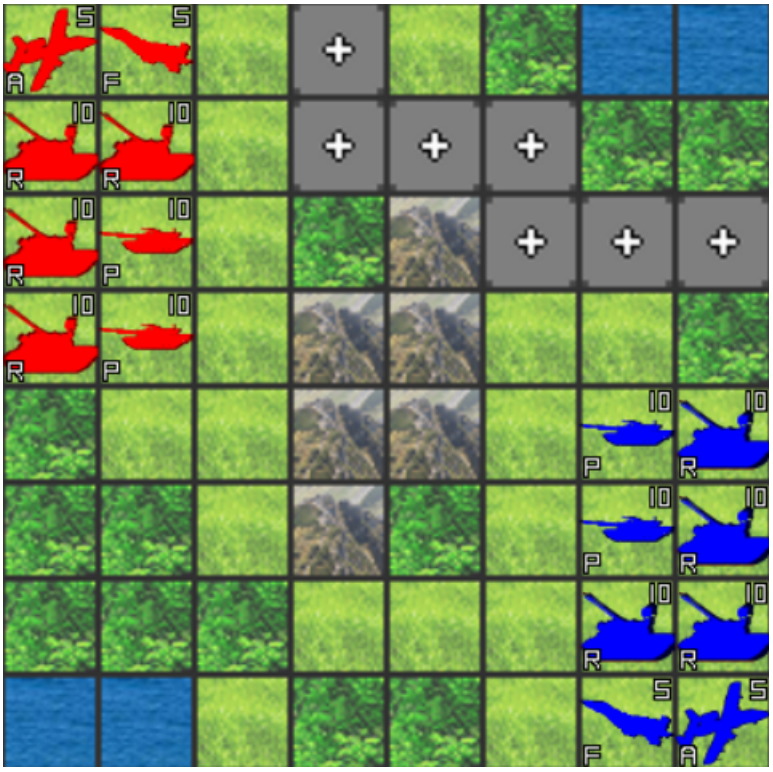

Figure 1: Starting TUBSTAP map (GPW_fujiki.tbsmap)

4.2 Results

Table 3: Experiment Results

Agent 1	Agent 2	Victor	Sample/Determinization Count	Iterations	Win Rate
PIMC	MO-ISMCTS	PIMC	All	All	1
PIMC	BS-MCTS	PIMC	All	All	1
IS-MCTS	BS-MCTS	Draw	All	All	0.5

In all experiments conducted, the PIMC agent is able to outperform the MO-ISMCTS and BS-MCTS agent in 100% of games played, regardless of iteration, determinization of simulation limits. The vast majority of matches were won by annihilation of the opposing team, with a small number won by possessing more units at the turn limit. The reasons for this are unclear. Previous research on the MCTS (including hidden information variants) indicates that one of the prime factors predicting agent performance is search depth[4]: the more iterations allowed to the algorithm,

the deeper it can search in the tree- this colloquially allows the algorithm to see more moves ahead, improving search results and overall game-playing ability. One of the supposed drawbacks of the PIMC is that it is incapable of searching very deeply compared to other MCTS variations due to most of its iterations being spread out over a number of determinizations. IS-MCTS and BS-MCTS notionally offer improved search performance by performing a search over a single tree instead, allowing these algorithms to search deeper and identify a better solution [4].

Such a significant result seems unlikely- the cause is likely to be an unidentified bug or other issue in the PIMC, MO-ISMCTS and/or BS-MCTS agent code. MO-ISMCTS and BS-MCTS are more complex algorithms than PIMC [4, 5], which makes implementation more difficult. For example, an earlier iteration of the BS-MCTS agent strongly favored turn-end actions over any others due to a bug in the sampling stage. When populating the state with enemy units, the algorithm sometimes attempted to illegally place units out-of-bounds- consequently, the resulting state possessed no enemy units, and as a result the algorithm found no utility in move or attack actions. In another example, an earlier iteration of the PIMC agent were known to have information leakage issues, where the supposedly random determinizations were accidentally derived from the ground truth. This meant the agent only sampled from determinizations that were extremely similar to reality, giving it an unfair advantage over other agents. These particular issues (among others) are believed to have been fixed, but other bugs of a similar nature may still exist, artificially boosting or hampering agent performance. This is supported by the relative performance of the BS-MCTS and IS-MCTS agents, which universally result in a draw at the turn limit. In both cases, it appears that the MO-ISMCTS agent and BS-MCTS agent fail to prosecute attack actions as often as PIMC, resulting in highly degraded performance, though the reason for this is unknown.

Regardless, it appears that the PIMC is able to choose actions more effectively than the MO-ISMCTS or BS-MCTS agents. Subjectively, the PIMC agent tends to mass units together while the MO-ISMCTS agent tends to spread them across the playable area, which allows them to be individually picked out and destroyed by the PIMC agent. It's unknown why the MO-ISMCTS tends to prefer this behavior- it may be that the MO-ISMCTS algorithm somehow rewards information gaining actions more. The BS-MCTS does not demonstrate this behavior, but like MO-ISMCTS appears to deprioritize attack actions, resulting in defeat. The MO-ISMCTS agent may search deeper than the PIMC agent, but increased search depth does not benefit performance in this particular domain. Due to the uncertainty involved, the ability of the PIMC to sample multiple determinizations may actually be a greater asset. PIMC is known to outperform IS-MCTS in sufficient stochastic domains [7], offering significant advantage in multi-action turn-based domains with hidden information. On the other hand, the dominance of PIMC over BS-MCTS (which features a prominent sampling phase) disputes this claim, providing further evidence for bugs in the implementation.

Assuming no issues with the specific implementation, another potential cause is poor inference and opponent modelling: MO-ISMCTS and BS-MCTS both make an assumptions for the determinization at the start of its turn. For the first turn, this implementation assumes the enemy has the same unit composition positions (appropriately mirrored across the map, as is standard in competitive scenarios), but as the game progresses these units and positions are likely to change drastically, and this determinization and the resulting information sets are increasingly unhelpful. Better opponent modelling or other domain-specific knowledge may alleviate this.

During simulations, agents were allowed to take as much time to compute their next actions as necessary given simulation, iteration and determinization counts.

While not a factor for this experiment, another impacting agent performance may be the complexity of the algorithms. The MO-ISMCTS and BS-MCTS agents took as long as 120 seconds to compute an action, 10-50x longer than the PIMC agent. This could greatly affect agent performance where computation is limited by time and not iteration, sampling, or determinization count. Both BS-MCTS and MO-ISMCTS are longer and more complicated algorithms than PIMC, requiring more operations per iteration to handle the sharing of information sets and multiple player trees. One of the proposed benefits of MO-ISMCTS is its ability to search a small number of trees to find a better solution than an algorithm that searches many trees shallowly), but that might not be the case here. Given a strict time budget, MO-ISMCTS and BS-MCTS may not be able to search deeply due to the increased number of instructions per iteration when compared to PIMC.

Additionally, one final finding supports previous research [4]: for PIMC, more computation time equates to improved performance. When playing against itself, the PIMC agent with the higher determinization and/or iteration count was the victor in every match. This result was expected; however, some research suggests that improved determinization/sample counts do not significantly affect performance so long as the values chosen are sufficiently large [7]. The improved performance of the higher determinzation/sample count agents is likely related to the complexity of the domain. A deep search tree in a single-action game may be considered a very shallow tree in a multi-action domain in terms of turn look-ahead; for this domain, it's possible that the values for determinization/iteration count can be increased significantly before seeing diminishing returns in performance.

The impact of iteration/sample counts for the IS-MCTS and BS-MCTS agents against themselves was inconclusive. Due to the extremely high computation time required by each, neither was able to complete a full round of simulations. For the

completed matches, the results were the same as the matches between the IS-MCTS and BS-MCTS agents, where both agents fail to prosecute attack actions effectively and fight to a draw at the turn limit. Previous research has used far higher increment counts for these algorithms [6, 5] and (like PIMC) higher iteration budgets may improve performance, but unfortunately the linear increase in computation currently makes testing these values infeasible.

V. Conclusions

Multi-action turn-based domains with hidden information are highly complex, requiring the use of advanced search algorithms. This project implemented three Monte Carlo Tree Search (MCTS)-based game-playing agents in TUBSTAP, an open-source multi-action strategy game, further modified to feature hidden information via fog-of-war. The relationship between the performance of the MCTS and the domain it is applied is not well understood- this project intended to apply variations of the algorithm to a multi-action turn-based game with hidden information to improve this understanding. Three agents- based on the Determinized/Perfect Information MCTS (PIMC), Multi-Observer Information-Set MCTS (MO-ISMCTS) and Belief-State MCTS (BS-MCTS) respectively - were compared against each other. However, the experimental results were inconclusive- the PIMC-based agent outperformed the others in 100% of all matches, an unlikely result strongly indicating an unknown bug with the search code. However, MO-ISMCTS and BS-MCTS are known to be more complex and more difficult to implement than PIMC, and one may wish to take the results of this project into account when selecting an algorithm in a practical application.

5.1 Future Work

Despite the experimental results, there are several avenues of future work that may be worth pursuing:

- **Investigate additional, domain-specific heuristics and other enhancements.** Better pruning methods, prior/expert knowledge and other heuristics are known to improve search performance in other domains. Better pruning methods can reduce the branching factor and allow for a deeper search, poten-

tially allowing an agent to identify better solutions. Move-ordering to prioritize attack actions has been implemented in TUBSTAP before with promising results [2]. Better inference models explicitly reduce non-locality in BS-MCTS [5]. This project did not utilize advanced heuristics like these, all of which have the potential to focus the search and improve game-playing performance.

- **Investigate search parameter adjustments**. This project only investigated the effect of iteration and sample/determinization count, but each of the tested algorithms has several different variables than can be modified to influence the search. A modified UCB constant can affect the balance of exploration and exploitation for PIMC and MO-ISMCTS, while different weight and belief adjustment coefficients can affect search for BS-MCTS. In the case of BS-MCTS, these coefficients can instead be implemented as functions of the the state, further influencing search behavior.

- **Investigate gameplay adjustments**. This project only tested agents using a single map and a single unit composition. While this approach allows for a fair comparison of the search algorithms, it is not necessarily representative of the domain as a whole. Different map layouts, unit compositions and visibility ranges may affect algorithm performance via indirect rewards- for example, larger maps may benefit agents that reward information-gaining actions over agents that don't.

- **Compare results when limited by computation time, as well as iteration/sample count**. MO-ISMCTS and BS-MCTS take far longer to compute actions than PIMC for the same number of samples/iterations. These algorithms are also able to search deeper in their respective search trees, but this may not be beneficial in a sufficiently complex domain like this one. Limiting

search by overall computation time may affect agent performance as well.

- **Investigate additional algorithms.** PIMC, MO-ISMCTS and BS-MCTS are not the only MCTS extensions for hidden information domains. Other algorithms discussed in Chapter 2 may offer improved performance in this domain. In particular, the Single Observer IS-MCTS (SO-ISMCTS) is worth investigating as a comparison for the MO-ISMCTS. Additionally, various learning algorithms may be beneficial- while BS-MCTS utilizes online learning, the domain is sufficiently complex that offline or deep-learning algorithms may offer improved speed and performance.

Appendix A. Algorithms

1.1 Monte Carlo Tree Search

Algorithm 3 General MCTS Approach

```
1: function MctsSearch(s_0)
2:     create root node v_0 with state s_0
3:     while Within computational budget do
4:         v_l ← TreePolicy(v_0)
5:         Δ ← DefaultPolicy(s(v_l))
6:         Backup(v_l, Δ)
7:     end while
8:     return a(BestChild(v_0))
9:
```

1.2 Negamax

Algorithm 4 Negamax

```
1: function Negamax(node, depth, color)                    ▷
2:     if depth = 0 or node is terminal then
3:         return Color* Heuristic value of node
4:     end if
5:     value := −∞
6:     for each child of node do
7:         value := max(value, −negamax(child, depth − 1, −color))
8:     end for
9:     return value
10:
```

1.3 MCTS UCT

Algorithm 5 MCTS with UCT

function UCTSEARCH(s_0)
  Create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow TreePolicy(v_0)$
    $\Delta \leftarrow DefaultPolicy(s(v_l))$
    $Backup(v_l, \Delta)$
  end while
  return $a(BestChild(v_0, 0))$
end function

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v is not fully expanded **then**
      return Expand(v)
    else
      $v \leftarrow BestChild(v, C_p)$
    end if
  end while
  return v
end function

function EXPAND(v)
  choose $a \in$untried actions from $A(s(v))$
  add a new child v' to v
  with $s(v') = f(s(v), a)$
  and $a(v') = a$ **return** v'
end function

function DEFAULTPOLICY(s)
  while v is non-terminal **do**
    Choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  end while
  return reward for state s
end function

function BESTCHILD(v, c)
  return Max ($v' \in$ children of v)) $Q(v')/N(v') + c\sqrt{2 \ln N(v)/N(v')}$
end function

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v
  end while
end function

1.4 Perfect Information/Determinized MCTS

Algorithm 6 Perfect Information / Determinized MCTS

function SEARCH(s_0)
  Create root node v_0 with state s_0
  while within computational budget **do**
    randomly select state $s_0 \in$ set of all states S
    $v_l \leftarrow TreePolicy(v_0)$
    $\Delta \leftarrow DefaultPolicy(s(v_l))$
    $Backup(v_l, \Delta)$
  end whilereturn $a(BestChild(v_0, 0))$
end function

function TREEPOLICY(v)
  while v is nonterminal **do**
    if $visnotfillyexpanded$ **then**
      return Expand(v)
    else
      $v \leftarrow BestChild(v, C_p)$
    end if
  end while
  return v
end function

function EXPAND(v)
  choose $a \in$untried actions from $A(s(v))$
  add a new child v' to v
  with $s(v') = f(s(v), a)$
  and $a(v') = a$ **return** v'
end function

function DEFAULTPOLICY(s)
  while v is non-terminal **do**
    Choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  end while
  return reward for state s
end function

function BESTCHILD(v, c)
  return Max ($v' \in$ children of v)) $Q(v')/N(v') + c\sqrt{2 \ln N(v)/N(v')}$
end function

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v
  end while
end function

1.5 MO-ISMCTS

Algorithm 7 Multi Observer IS-MCTS (MO-ISMCTS)

procedure MO-ISMCTS($[s_0]^1, n$) ▷ This is a test
    for each player $p_0...p_k$ create a single-node tree with root v_i^0 (representing $[s_0]^1$ from player i's viewpoint)
    for n iterations **do**
        choose a determinzation d at random from $[s_0]^1$ and use only nodes/actions compatible with d for this iteration.
        (SELECTION)
        repeat
        descend all trees in parallel using a bandit algorithm on player p's tree whenever p is about to move
        until
        nodes $v_0...v_p...v_k$ are reached in trees $0...k$ respectively, player p is about to move at node v_p, and some action from v_p leads to a player p information set which is not currently in player p's tree
        or until
        v_p is terminal

        (EXPANSION)
        if v_p is nonterminal **then**
            Choose at random an action a from node v_p that is compatible with d and does not exist in the player p tree
            for each player $i..k$ **do**
                If there is no node in player i's tree corresponding to action a at node v^i, then add such a node
            end for
        end if

        (SIMULATION)
        Run a simulation from v_p to the end of the game using determinization d (starting with action a if v_p is nonterminal)

        (BACKPROPAGATION)
        for each node u_i visited during this iteration for all players i **do**
            update u_i's visit count and simulation reward
            for each sibling w_i of u_i that was available for selection when u_i was selected (including u_i) **do**
                Update w_i's availability count
            end for
        end for
        return an action from $A[s_0]^1$ such that the number of visits to the corresponding child of v_0^1 is maximal
    end for

1.6 BS-MCTS

Algorithm 8 Belief-State MCTS (BS-ISMCTS)

Require B_{root}, maximal samplings T, maximal iterations S.

function BS-MCTS(B_{root})
  $t \leftarrow 1$
  while $t \leq T$ **do**
    $\gamma \leftarrow$Sampling(B_{root})
    $s \leftarrow 1$
    while $s \leq S$ **do**
      $R \leftarrow$Search(γ,B_{root})
      $N(\gamma) \leftarrow N(\gamma) + 1$
      $s \leftarrow s + 1$
    end while
    $t \leftarrow t + 1$
  end while
end function

function EXPANSION(γ, B)
  $N(\gamma) \leftarrow 0$
  for all $a \in A(\gamma)$ **do**
    if $B \cdot a$ is not in the tree **then**
      add $B \cdot a$ to the tree
    end if
    if $\gamma \cdot a$ not in $B \cdot a$ **then**
      add $\gamma \cdot a$ to $B \cdot a$
      $N(\gamma, a) \leftarrow 0$
      $U(\gamma, a) \leftarrow 0$
    end if
  end for
end function

function SAMPLING(B_{root})
  generate new γ
  add $\gamma to B_{root}$
end function
function SEARCH(γ, B)
  if $N(B)0$ **then**
    $R \leftarrow$Simulation(γ)
    return R
  end if
  if γ has no children **then**
    $Expansion(\gamma, B)$
  end if
  $N(\gamma) \leftarrow N(\gamma) + 1$
  action $a \leftarrow$Selection(γ, B)
  $R \leftarrow$Search($\gamma \cdot a, B \cdot a$)
  $N(\gamma, a) \leftarrow N(\gamma, a) + 1$
  $U(\gamma, a \leftarrow U(\gamma, a) + \frac{1}{N(\gamma,a)}[R - U(\gamma, a)]$
  return R
end function

Appendix B. Tables

Table 4: Unit Visibility Range

Unit Type	Visibility Range (tiles)
Fighter Jet	5
Attack Jet	5
Tank	2
Artillery	2
Anti-Air	2
Infantry	1

Table 5: Historical Iteration Budgets

Paper	Iteration Budget
Whitehouse [4]	2500, 5000, 10000, 15000
Fernandes [6]	500, 2500, 10000, 20000, 30000, 40000, 50000
Wang, et al. [5]	100,200,400,800,1000,2000,2500,5000,20000,40000,80000,100000,200000
Ihara, et al. [23]	1000

Table 6: Experiment List

Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	1	1000	1000
MO-ISMCTS	N/A	1000	1000
BS-MCTS	10	100	1000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	5	1000	5000
MO-ISMCTS	N/A	5000	5000
BS-MCTS	10	500	5000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	10	1000	10000
MO-ISMCTS	N/A	10000	10000
BS-MCTS	10	1000	10000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	20	1000	20000
MO-ISMCTS	N/A	20000	20000
BS-MCTS	20	1000	20000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	10	100	1000
BS-MCTS	10	100	1000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	50	100	5000
BS-MCTS	50	100	5000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	100	100	5000
BS-MCTS	100	100	5000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
PIMC	1	100	5000
PIMC	5	1000	5000
PIMC	10	1000	10000
PIMC	20	1000	20000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
MO-ISMCTS	N/A	1000	1000
MO-ISMCTS	N/A	5000	5000
MO-ISMCTS	N/A	10000	10000
MO-ISMCTS	N/A	20000	20000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
MO-ISMCTS	N/A	1000	1000
MO-ISMCTS	N/A	5000	5000
MO-ISMCTS	N/A	10000	10000
MO-ISMCTS	N/A	20000	20000
Agent Type	Samples/Determinizations	Iterations	Total Iterations
BS-MCTS	10	100	1000
BS-MCTS	10	500	5000
BS-MCTS	10	1000	10000
BS-MCTS	20	1000	20000

Table 7: Experimental Results

Agent 1	Agent 2	Victor	Sample/Determinization Count	Iterations	Win Rate
PIMC	MO-ISMCTS	PIMC	All	All	1
PIMC	BS-MCTS	PIMC	All	All	1
IS-MCTS	BS-MCTS	Draw	All	All	0.5

Bibliography

1. Matthew Caffrey Jr. Toward A History-Based Doctrine For Wargaming. Technical report, Air and Space Power Journal, Maxwell AFB AL, 2000.

2. Tsubasa Fujiki, Kokolo Ikeda, and Simon Viennot. A Platform For Turn-based Strategy Games, With A Comparison Of Monte-Carlo Algorithms. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 407–414. IEEE, 2015.

3. Russel Stuart and Norvig. *Artificial Intelligence: A Modern Approach (3rd Ed)*, volume 4. 2010.

4. Daniel Whitehouse. *Monte Carlo Tree Search For Games With Hidden Information and Uncertainty*. PhD thesis, University of York, 2014.

5. Jiao Wang, Tan Zhu, Hongye Li, Chu Husan Hsueh, and I. Chen Wu. Belief-state Monte-Carlo Tree Search for Phantom Go. *IEEE Transactions on Computational Intelligence and AI in Games*, 10(2):139–154, 2017.

6. Pedro Ricardo Oliveira Fernandes. Framework for Monte Carlo Tree Search-related strategies in Competitive Card Based Games. 2016.

7. Tamirlan Omarov, Hamna Aslam, Joseph Alexander Brown, and Elizabeth Reading. Monte Carlo Tree Search for Love Letter, 2018.

8. Jahn-Takeshi Saito. Solving Difficult Game Positions. 2010.

9. Louis Victor Allis et al. *Searching For Solutions In Games And Artificial Intelligence*. Ponsen & Looijen Wageningen, 1994.

10. Jonathan Schaeffer. Search Ideas in Chinook. *Games in AI Research*, pages 19–30, 2000.

11. Wilfrido Perez Cutwright. Developing Artificial Intelligence Agents for a Turn-Based Imperfect Information Game. Master's thesis, Liberty University, 2019.

12. Aditya Bhatt, Scott Lee, Fernando de Mesentier Silva, Connor W Watson, Julian Togelius, and Amy K Hoover. Exploring The Hearthstone Deck Space. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, pages 1–10, 2018.

13. Naoyuki Sato and Kokolo Ikeda. Three Types Of Forward Pruning Techniques To Apply The Alpha Beta Algorithm To Turn-Based Strategy Games. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

14. Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Senior Member, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 2012.

15. Sylvain Gelly and Yizao Wang. Exploration Exploitation In Go: UCT for Monte-Carlo Go. In *NIPS: Neural Information Processing Systems Conference On-line Trading of Exploration and Exploitation Workshop*, 2006.

16. Niels Justesen, Tobias Mahlmann, Sebastian Risi, and Julian Togelius. IEEE Transactions on Computational Intelligence and AI in Games Playing Multi-Action Adversarial Games: Online Evolutionary Planning Versus Tree Search. Technical report.

17. Peter I. Cowling, Colin D. Ward, and Edward J. Powley. Ensemble Determinization In Monte Carlo Tree Search For The Imperfect Information Card Game Magic: The Gathering. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):241–257, 2012.

18. Hiroyuki Ihara, Shunsuke Imai, Satoshi Oyama, and Masahito Kurihara. Implementation and Evaluation of Information Set Monte Carlo Tree Search for Pokémon. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2182–2187. IEEE, 2018.

19. Hendrik Baier and Mark HM Winands. MCTS-Minimax Hybrids With State Evaluations. *Journal of Artificial Intelligence Research*, 62:193–231, 2018.

20. James Goodman. Re-determinizing MCTS in Hanabi. *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2019-Augus, 2019.

21. Moshe Bitan and Sarit Kraus. Combining Prediction Of Human Decisions With ISMCTS In Imperfect Information Games. *arXiv preprint arXiv:1709.09451*, 2017.

22. Jay Burmeister and Janet Wiles. The Challenge Of Go As A Domain For AI Research: A Comparison Between Go And Chess. In *Proceedings of Third Australian and New Zealand Conference on Intelligent Information Systems. ANZIIS-95*, pages 181–186. IEEE, 1995.

23. Hiroyuki Ihara, Shunsuke Imai, Satoshi Oyama, and Masahito Kurihara. Implementation and Evaluation of Information Set Monte Carlo Tree Search for Pokémon. Technical report.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE

APPLICATION OF THE MONTE-CARLO TREE SEARCH TO MULTI-ACTION TURN-BASED GAMES WITH HIDDEN INFORMATION

5a. CONTRACT NUMBER

5b. GRANT NUMBER

20G194

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Connor M. Pipan

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-072

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RHA
Building 441
WPAFB OH 45433-7765
DSN 785-0215, COMM 937-255-0215
Email: winston.bennett@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RH

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Traditional search algorithms struggle when applied to complex multi-action turn-based games. The introduction of hidden information further increases domain complexity. The Monte-Carlo Tree Search (MCTS) algorithm has previously been applied to multi-action turn-based games, but not multi-action turn-based games with hidden information. This thesis compares several Monte Carlo Tree Search (MCTS) extensions (Determinized/Perfect Information Monte Carlo, Multi-Observer Information Set MCTS, and Belief State MCTS) in TUBSTAP, an open-source multi-action turn-based game, modified to include hidden information via fog-of-war.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Captain Connor M. Pipan, AFIT/ENG
U	U	U	UU	54	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636; connor.pipan@afit.edu