Incentivizing Information Gain for MCTS in Hidden Information Multi-Action Games

REPORT

Nathan Lervold, Second Lieutenant, USAF

AFIT-ENG-MAS-22-S-026

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MAS-22-S-026

Incentivizing Information Gain for MCTS in Hidden Information Multi-Action Games

REPORT

Presented to the Faculty

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 Cyber Systems

Nathan Lervold, BSCS

Second Lieutenant, USAF

September 15, 2022

AFIT-ENG-MAS-22-S-026

Incentivizing Information Gain for MCTS in Hidden Information Multi-Action Games

REPORT

Nathan Lervold, BSCS
Second Lieutenant, USAF

Committee Membership:

Gilbert L. Peterson, Ph.D
Chair

Lt Col. David W. King, Ph.D
Member

AFIT-ENG-MAS-22-S-026

Abstract

Hidden Information is a central mechanic in games like the Resistance or wargaming with fog of war, adding an extra layer of complexity for search algorithms. Monte Carlo Tree Search (MCTS) has gained much notoriety given its success in searching complex domains such as Go. Extensions to MCTS allow it to perform well in hidden information games such as Bridge, Kriegspiel (Chess), and Magic the Gathering. These MCTS extensions however fail to consider information gain, an important aspect of multi-action hidden information games as initial actions inform sequential decisions. This report proposes an information gain incentive function and a risk function to offset the risk of information gain. We implement the information gain incentive and risk functions into MCTS variants ISMCTS and PIMCTS which are then tested in the multi-action hidden information game TUBSTAP. Overall testing demonstrates promising results, but lack of consistency makes it largely inconclusive. Current implementation of the information gain incentive has flaws, and we offer a more effective approach.

Table of Contents

List of Figures

List of Tables

Incentivizing Information Gain for MCTS in Hidden Information Multi-Action Games

I. Introduction

Wargaming is a key part of course of action (COA) analysis, allowing a commander to visualize and test a plan in detail. COA analysis is a step in the military decision-making process, a process used to create orders from the reception of a mission. War gaming is often represented as a turn-based strategy or tactical game (such as Silver Botonet, Flashpoint Campaigns, Main Battle Tank, and Axis and Allies) [1].

Artificial Intelligence (AI) has been applied to wargaming to propose changes to a preexisting COA [2] or develop an initial COA [3]. Additionally, AI can create competitive game agents for individual play. Wargaming can require much coordination and time to set up and play as a board game, especially if trying to employ aspects such as fog of war which may require a game manager. Digitization of wargames with competitive AI agents allows for much more training and repetition of COAs as less coordination and time are required.

Wargames employ many game elements that greatly increase the complexity beyond a game of checkers or chess. These may include fog of war (hidden information), taking multiple actions each turn (multi-action turns), having different capabilities for each unit (such as unit health), or having a large board size. Additionally, war games utilize a variety of maps and starting pieces to accommodate different scenarios. This with the complex interaction between each unit and its environment makes developing AI agents difficult.

TUBSTAP, the TUrn Based STrategy Academic Package, [4] is a multi-action turn-based strategy game that encompasses many complexities of wargaming. TUB-STAP includes map customization, various unit types and compositions, fog of war (by extension of [5]), and terrain that imposes restrictions and bonuses. In a TUB-STAP game, two teams compete to eliminate all units of the opposing team.

Monte Carlo Tree Search (MCTS) has proven to be very successful in games of increasing complexity such as Go [6]. Additionally, it has been successfully applied to multi-action games [4] and games with hidden information such as Bridge [7], Kriegspiel [8] [9], and Magic [10]. One shortcoming of MCTS applied to multi-action hidden information games, is that it does not consider taking actions for the benefit of gaining information, such as moving a unit to uncover fog of war to give future moves more certainty.

1.1 Research Hypothesis

The hypothesis is that adding an information gain incentive with a risk function to offset information gain inherent risk will improve MCTS hidden information variants in multi-action hidden information games. This report proposes the following research questions:

- Does applying an information gain incentive at MCTS selection phase improve MCTS performance?
- Does applying a risk function to weight determinizations help offset the risk of information gain?
- How do the information gain incentive and risk function perform together and separately?
- How does applying risk and hidden information modification behave differently when applied to MCTS variants ISMCTS and PIMCTS?

1.2 Methodology

Chapter III first explains how to prune impossible determinizations in TUBSTAP with the use of a probability map in Section 3.1. Section 3.1.1 defines the information gain incentive function, explains design choices, and applies it to MCTS variants. The information gain incentive function determines the value of information gained by examining a unit's gained visibility and how this visibility uncovers information on the pmap. The value of the function weights some nodes higher during the selection phase of MCTS variants. Section 3.1.2 defines the risk function, explains design choices, and applies it to MCTS variants. The risk function weights the pmap to guide determinizations to be riskier.

1.3 Results

Chapter IV first designates function parameters, defines MCTS variants, displays test maps, and explains the testing process in Section 4.1. Section 4.2 displays and highlights the results of the experiment. Overall, the information gain function with risk function did improve performance in some situations. When combined with ISMCTS and tested on a larger map, it instead performed worse, demonstrating inconsistency. Lastly, Section 4.3 analyzes the results, explores why this inconsistency may have occurred, and possible solutions. Other key findings include that the integration of a pmap to eliminate impossible determinizations greatly increases performance. Additionally, ISMCTS outperforms PIMCTS on all maps when using the pmap integration.

1.4 Document Overview

This document is organized as follows. Chapter II explains the rules of TUBSTAP, discusses how multi-action and hidden information elements apply to game tree searches, and defines the MCTS variants PIMCTS and ISMCTS. Chapter III details the risk and information gain incentive functions and their integration into PIMCTS and ISMCTS. Chapter IV presents the results of the risk and incentive functions applied to PIMCTS and ISMCTS when competing on three maps. Finally, Chapter V discusses the conclusions drawn from the results.

II. Background and Literature Review

TUBSTAP with fog of war has a very complex search space as a hidden information and multi-action game. While MCTS is suitable for complex search environments, it requires extensions to handle hidden information and requires a definitive game tree structure to handle multi-action games.

This chapter explains the rules of TUBSTAP, defines hidden information, explains MCTS and its variants PIMCTS and ISMCTS in the context of game tree search, and gives background on game tree search methods applied to both multi-action games and hidden information games. Additionally, it highlights the unique considerations of games when hidden information and multi-action aspects are handled separately and together.

2.1 TUBSTAP Rules

TUBSTAP game mechanics are based on Nintendo's 2001 game Advanced Wars. TUBSTAP is a two-player game, each player controls a team of units. During a turn, each player moves their units one at a time until all units are moved. Winning occurs when one player eliminates all units of the opposing team. Teams fight on a tile map as shown in Figure 1. The game has the following rules:

- Each tile has a terrain type that affects movement and defensive bonuses.
- There are six types of units: Attack Jet (A), Fighter Jet (F), Infantry (I), Anti-Air (R), Tank (P), and Artillery (U)
- Each unit has a list of unique attributes. Table 1 displays the attack power that each unit type has against another unit type. Units can attack adjacent units (1 unit away), except for artillery which can attack units at 2 and 3 tiles away.

- Damage is calculated as follows:

$$damage = \frac{attackpower \times attackHP}{10 + terrainbonus \times defenseHP} \tag{1}$$

- Each unit type has a unique movement range. Units can move any number of tiles within that range. Moving over a tile costs 1 or 2 depending on terrain and unit type. Additionally, some terrain blocks movement of certain unit types.

- On one player's turn they perform an action for all their units in any order. An action consists of a move and possibly an attack. Thus, if a player has 6 units, they take 6 actions.

This research adds the rule of unit vision implemented by [5] that functions as a

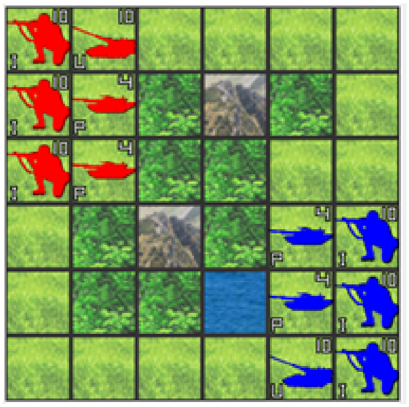

Figure 1: Example TUBSTAP Map.

Table 1: Attack Power Matrix.

	F	A	P	U	R	I	S
F	55	65	0	0	0	0	0
A	0	0	105	105	85	115	105
P	0	0	55	70	75	75	75
U	0	0	60	75	65	90	70
R	70	70	15	50	45	105	55
I	0	0	5	10	3	55	35
S	0	0	10	20	20	55	55

Table 2: Unit Vision and Movement.

Unit Type	A	F	R	I	P	U	S
Vision	3	3	2	2	2	1	4
Movement	7	6	5	4	5	2	6

Fog of War. Each unit's vision value is equal to the number of tiles it can see from its current position in any direction (Table 2). A player's vision is equal to the super-set of each unit's visible tiles. Enemy unit locations are only known when they are within the player's vision.

- Enemy units can only be seen on tiles that are in the player's vision.
- A unit must decide what it is attacking before moving. Thus, enemy units cannot be attacked by a unit unless they were visible at the beginning of that unit's action. Any enemy units revealed by a friendly unit's movement can then be attacked by the next friendly unit.
- Each player starts the game with knowledge of the number of types of units.

2.2 Hidden Information Game Search Fundamentals

Powly, et al. [11] define imperfect information games as having states that are partially observable with different observations for different players. Perfect information games have states that are fully observable to all players. The non-observable information in the state is classified as hidden information.

TUBSTAP with Fog of War becomes a hidden information game. Each unit has a certain sight range and anything outside of the unit's sight range is unknown. Hidden information exists in Texas Holdem as a player cannot observe an opponent's hand. This hidden information is different from stochastic, or chance, elements in a game. For example, in Texas Holdem each card dealt on the board is a random card from the deck. Stochastic elements, such as a random card draw, can be classified as occurring with a designated probability. Hidden information may be gathered, leaked, and inferred. Table 3 classifies games according to hidden information and stochasticity. TUBSTAP with Fog of War is a deterministic game with imperfect information.

Table 3: Contrast of Imperfect Information (contains hidden information) and Stochastic Games.

	Deterministic	Stochastic
Perfect Information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect Information	TUBSTAP, Wumpus World	Bridge, Poker, Scrabble

Hidden information contributes to increased complexity and branching factor. Unknown space can be represented as a list of possible determinations. Consider a TUBSTAP board where there are 4 unseen enemy units and 20 unseen tiles. The number of possible determinations is $C(20, 4) \times 4!$ or 116,280 determinations. The branching factor of the game is then multiplicative of the number of determinations.

Determinations can be guided by beliefs of the opponent. The opponent behaves according to a policy allowing inference on which determinizations are more likely. The better a player's beliefs of the opponent, the fewer determinations needed to make an optimal decision. The simplest way to model an opponent would be to consider all possible determinizations with an equal probability of occurring.

Lastly, hidden information can be reduced through certain information-gaining actions. In TUBSTAP, by moving a unit closer to unseen tiles, the player can learn whether tiles within visual range contain an enemy or are empty. Taking an action that is suboptimal by itself based on the known information, may lead to more informed and thus better subsequent actions. TUBSTAP being a multi-action game only enhances the importance of information gain as it informs subsequent actions. By searching through a sampling of possible determinizations, a player can determine the average best move from the perspective of a single unit. This does not factor in the utility of information gain that can benefit all other units.

2.3 Game Tree Search

A search tree is used to represent the possible states of a game. The board state at the beginning of a player's turn is the root of the tree with a depth of 0. The root is expanded by applying each possible action to the root to generate the children's nodes with a depth of 1. At each depth d, node expansion generates children at a depth of $d + 1$ until a terminal node is reached. A terminal node has no possible actions to be taken and evaluates to a win or loss for the player. Expanding all nodes in the search tree to terminal nodes generates all possible legal positions from the root. The branching factor b is the number of children at each node. For example, a binary tree has a branching factor of two. However, the branching factor for game trees varies from node to node, making it useful to calculate an average branching factor. The number of nodes in the tree can then be estimated as b^d. The purpose of the game tree is to determine the best action to take from the root node. Action candidates are the set of actions that lead to the root children at $d = 1$.

Minimax Search is a tree search method that returns an optimal action in a deterministic adversarial game [12]. A minimax game tree as shown in Figure 2 alternates between player nodes and opponent nodes at each ply. Minimax search assumes that each player plays optimally. Thus, opponent nodes (minimizing nodes) always evaluate to the minimum value of thier children. And player nodes (maximizing nodes) always evaluate to the maximum value of their children. An evaluation function determines the value of terminal nodes, for example, a 1 for a win, 0 for a draw, and a -1 for a loss.

Expectiminimax can be used for an adversarial game with hidden information or stochastic elements [13]. Expectiminimax uses chance nodes to account for hidden and stochastic elements as shown in Figure 3. Each child of a chance node is a possible outcome. They can also be used with hidden information where each child is

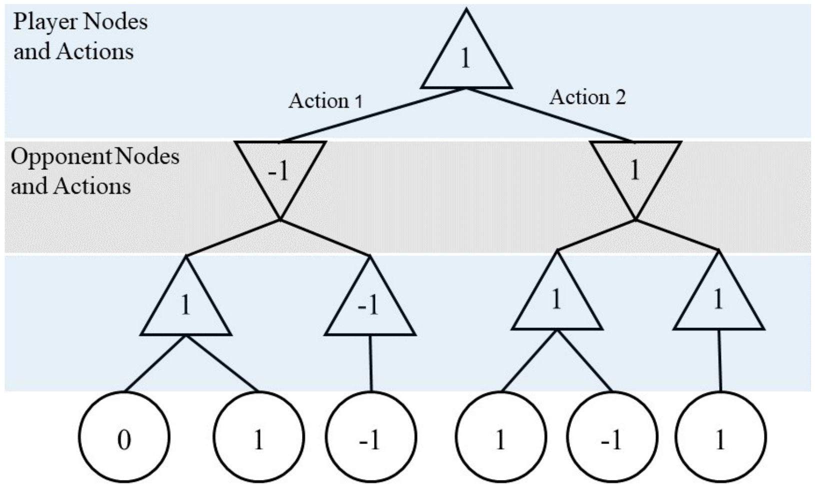

Figure 2: Minimax Game Tree. Maximizing nodes as triangles, minimizing nodes as upside-down triangles. Terminal nodes as circular nodes with an evaluation of win (1), loss (-1), or tie (0). Each line is a possible action that leads to a new possible state. The player will select Action 2 from the action candidates with a value of 1.

a possible determinization of the opponent. Each chance node assigns a probability to its child node; the total value of a chance node is then the expected value of its children. Maximizing and minimizing nodes behave the same.

Minimax and Expectiminimax find the optimum move for the player assuming

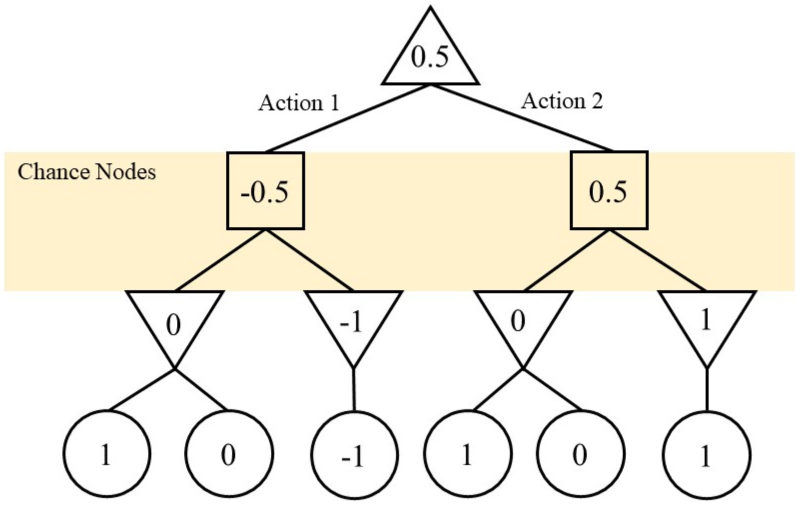

Figure 3: Expectiminimax Game Tree. Square nodes are chance nodes. Assuming an equal chance of all child nodes, each chance node takes the sum of the children divided by the number of children. The player will take Action 2 with a value of 0.5.

that the opponent also behaves optimally. However, for games with large branching factors, these search methods by themselves are impractical due to the exponential growth of the game tree. Heuristic pruning techniques such as Alpha-Beta Pruning can limit the number of nodes searched. For example, consider a minimizing node choosing between two maximizing nodes A and B. Node A is fully expanded and evaluates to 3. If any children of node B are discovered to be greater than 3, node B's unsearched children are pruned, since the minimizing node will always choose node A. Alpha-Beta minimax uses backward pruning, still guaranteeing an optimal solution [12]. Forward pruning also reduces tree complexity at the risk of overlooking an optimum solution. It attempts to guide the search toward a "good" solution by employing various heuristic methods. The next section discusses some forward pruning methods in the context of MCTS.

2.3.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an anytime stochastic algorithm that searches the game space asymmetrically, focusing on the most promising nodes [14]. MCTS has proven to perform well in games with large branching factors such as Go. Each iteration of MCTS completes four phases: Selection, Expansion, Simulation, and Backpropagation.

Selection: Starting at the root, MCTS selects child nodes according to a selection policy until it has reached a leaf node. The Upper Confidence Bounds Applied to Trees (UCT) algorithm Equation (2) is the most used selection policy for MCTS. [15]

$$UCT_i = \frac{w_i}{n_i} + \alpha\sqrt{\frac{\ln(N_i)}{n_i}} \tag{2}$$

w_i is the number of wins of node i. n_i is the number of visits to that node. N_i is the number of visits made to the node's parent. The UCT algorithm balances exploitation and exploration by exploiting the most promising nodes (those with the highest win ratio $\frac{w_i}{n_i}$) as well as exploring those which have not been visited often. Exploration is guided by the constant α. When examining a node that has never been visited before ($n_i = 0$), this node will always be selected as $\lim_{n_i \to 0} \sqrt{\frac{\ln(N_i)}{n_i}} = \infty$.

Expansion: Once a leaf node has been reached, a random successor state is added as a child node to this leaf node. Simulation is then conducted on this child node.

Simulation: Simulation is done through a rollout. A rollout selects children of the node until a terminal node is reached. A random rollout that selects random children is commonly used. The resulting win or loss of the simulation is then used during backpropagation.

Backpropagation: Starting at the child node added to the tree in the expansion phase, MCTS updates W_i and n_i of the node. This is repeated on each parent node until the root is reached. If the result of the simulation was a win, then W_i is incremented by 1. n_i is always incremented by one.

Although it is an anytime algorithm, if run with infinite time, the MCTS tree will converge to a full minimax tree.

Algorithm 1 displays the pseudocode for MCTS with the additional definitions. $A(s)$ is the set of all possible actions from the state s. $A(s) = \emptyset$ if s is terminal. $a(s)$ is the action that was taken to get to this state. $C(s)$ is the set of possible children from state s. $p(s)$ is the parent of state s. Using the terminology from Equation (2), n_s refers to visits of node s, W_s refers to wins.

MCTS can be adapted to account for hidden information by sampling possible determinations of the unknown space. Two hidden information MCTS variations: Perfect Information MCTS (PIMCTS) [12] and Information Set MCTS (ISMCTS) [10].

Algorithm 1: MCTS

```
Data: current state of the game
Result: action from candidate actions
1  initialize $root$ as the current state of the game;
2  while $time\ is\ left\ or\ n_{root} < MAX\_ITER$ do
3      $r = root$;
       // SELECT
4      while $A(r) \neq \emptyset$ do
5          $r$ = Select child with max score from $C(r)$;
       //EXPAND
6      if $\mid A(r) \mid > 0$ then
7          $a$ = select random action from $A(r)$;
8          $child$ = execute action $a$ on deepcopy of $r$ add $child$ as a child to $r$;
       //SIMULATE
9      $sim$ = deepcopy of $r$;
10     $result$ = perform a rollout on $sim$ according to the policy until a terminal
         node is reached;
       //BACKPROPOGATE
11     while $r! = null$ do
12         $n_r$ += 1 ;
13         $W_r+ = result$;
14         $r = p(r)$;
15 $best$ = select child with max UCT score from $C(root)$;
16 return $a(best)$;
```

2.3.2 Perfect Information MCTS

PIMCTS creates x determinizations from the game state, then performs a separate MCTS search on each determinization. The result of each determinization is stored in a candidate actions list. Once complete, the action with the highest cumulative UCT score is chosen.

PIMCTS has the shortcomings of strategy fusion and non-locality [16]. Strategy fusion is due to each determinization being treated as a perfect information game with no hidden information. Thus, the AI chooses an action based on unavailable information. Non-locality occurs when PIMCTS samples determinations with uniform probability. In reality, some states are more likely than others since they have a rational opponent.

Long, et al. [17] identified three features, leaf correlation, bias, and disambiguation that largely affected the success of PIMCTS. Leaf correlation is the probability that child nodes have the same payoff value. Bias is the probability that a game favors one player. Disambiguation is the speed that hidden information is revealed. PIMCTS performs better with higher leaf correlation and when disambiguation avoids the extremes of very fast or very slow. More extreme bias resulted in better performance if the correlation was medium to low. PIMCTS is likely a good candidate for TUBSTAP due to having a high leaf correlation, although disambiguation can largely depend on map size and the number of units. Pseudocode for PIMCTS is displayed in Algorithm 2.

2.3.3 Information Set MCTS

ISMCTS creates a new determinization d with each iteration. All determinizations are included in singular tree T_R and each node is a part of an information set. ISMCTS conducts MCTS on d, creating a search tree T_d, and adds the result to T_R. To do this, it selects nodes in T_R that exist in T_d and descends both trees in parallel. Then it expands a leaf node in T_R by adding an unexplored child node from T_d.

By aggregating the results of each determinization into a singular tree, ISMCTS overcomes the problem of strategy fusion. However since all determinizations are considered with the same probability, ISMCTS still suffers from non-locality. The effects of correlation, disambiguation, and bias likely have the same impact on ISMCTS as on PIMCTS since they both use determinizations. Psuedocode for ISMCTS is displayed in Algorithm 3.

Algorithm 2: PIMCTS

```
Data: current state of the game, max number of determinations
      $MAX\_DETER$
Result: action from candidate actions
1  initialize $root$ as the current state of the game;
2  $candidateActions = \emptyset$;
3  for $i = 0$ to $MAX\_DETER$ do
4      while $time\ is\ left\ or\ n_{root} < MAX\_ITER$ do
5          $r$ =create random determination from $root$;
           // SELECT
6          while $A(r) \neq \emptyset$ do
7              $r$ = Select child with max score from $C(r)$;
           //EXPAND
8          if $\mid A(r) \mid > 0$ then
9              $a$ = select random action from $A(r)$;
10             $child$ = execute action $a$ on deepcopy of $r$ add $child$ as a child to $r$;
           //SIMULATE
11         $sim$ = deepcopy of $r$;
12         $result$ = perform a rollout on $sim$ according to the policy until a
            terminal node is reached;
           //BACKPROPOGATE
13         while $r! = null$ do
14             $n_r$ += 1 ;
15             $W_r+ = result$;
16             $r = p(r)$;
17         $best$ = select child with max UCT score from $C(root)$;
18         if $a(best) \in candidateActions$ then
19             add the UCT scores of the actions together;
20         else
21             $candidateActions \cup a(best)$;
22 return $a(best)$;
```

Algorithm 3: ISMCTS

```
Data: current state of the game
Result: action from candidate actions
1  initialize $root$ as the current state of the game;
2  while $time\ is\ left\ or\ n_{root} < MAX\_ITER$ do
3      $s$ = make random determination from root;
4      $r = root$;
       // SELECT
5      while $A(r) \neq \emptyset\ and \mid A(r) \cup A(s)! = \mid A(r) \mid$: do
6          $n$ = Select child with max UCT score from $C(s) \cap C(r)$;
7          Execute action $a(r)$ on $s$;
       //EXPAND
8      $untried = A(r) \cup A(s) - A(r)$;
9      if $\mid untried \mid > 0$ then
10         $u$ = select random action from $untried$;
11         execute action $u$ on $s$;
12         add $s$ as a child to $r$;
13         $r = s$
       //SIMULATE
14     $sim$ = deepcopy of $r$;
15     $result$ = execute actions on $sim$ according to the policy until a terminal
        node is reached;
       //BACKPROPOGATE
16     while $r! = null$ do
17         $n_r$ += 1 ;
18         $W_r + = result$;
19         $r = p(r)$;
20 $best$ = select child with max UCT score from $C(root)$;
21 return $a(best)$;
```

2.4 Multi-action Game Tree Search

Nodes in a multi-action game tree can either represent a single action (partial turn) or a complete action sequence (full turn) as displayed in Figure 4.

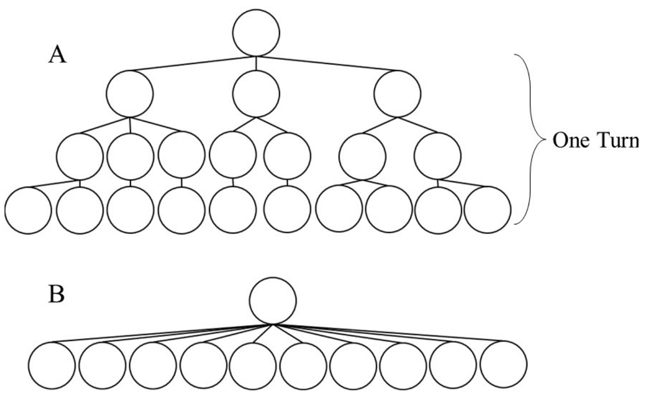

Figure 4: Two methods of constructing a Multi-Action Tree. Each node is an action (A) vs each node is a set of all actions (B).

Evolutionary MCTS [18] uses tree search where each node is a complete action sequence represented as a genome. Edges then represent a mutation of a single action of the genome. Additionally, Fujiki, et al. [4] uses Depth Limited Monte Carlo (DLMC) with nodes as an entire action sequence. DLMC creates m random samples from the list of all possible actions, then completes n simulations to a limited depth of d on each possible action. The state at the end of the simulations is evaluated by a static evaluator to give each sample a score. This benefits from easier state evaluation since total action sequences are used for nodes. Additionally, DLMC improves the quality of the m random samples through Attack Action Search (AAS). AAS examines l actions, exhausting all attack actions before including movement actions in the total for l. AAS then performs each action on the current state and then uses an evaluation function to determine the best action. DLMC + AAS then includes m random samples and the resulting best attack action from AAS before simulation.

When multi-action is combined with hidden information, representing each node as a complete action sequence leads to information loss. This is because each partial action sequence can uncover hidden information that informs the next partial action. However, nodes as a partial action sequence can decrease ply depth searched since simulation and determinations must be made at each action. Simulation and determinization computations may be repetitive if no information is gained between actions. To maintain information gain advantage, each node represented in PIMCTS and ISMCTS for testing in this paper will represent a single action. Additionally, the use of evolutionary algorithms and DLMC + AAS seem disadvantageous for multi-action games with hidden information given their tree structure where each node is a full turn.

Strategies applied to multi-action MCTS often include various heuristic pruning methods. Pruning is already a popular method to apply to MCTS outside of multi-action games for large search spaces [19][20]. The additional complexity provided by multi-action games only increases the value of limiting the tree space to obtain a deeper search. DLMC + AAS greatly reduces the TUSBTAP search space by eliminating a large portion of possible actions, essentially limiting the branching factor to m.

Sato and Ikeda [21] present three forward pruning techniques applied to TUBSTAP: Fixing the movement order of units, applying selective unit action generation, and applying limited unit actions. Selective unit action generation groups attack and movement actions by certain criteria, such as target unit, and only selects one action per group. Lastly, limited unit action performs a smaller tree search on only the most influential units (ignoring units with low HP or far out of range of combat). These forward pruning techniques improved TUBSTAP agents by allowing deeper searches on the tree focusing on more promising nodes.

Progressive widening or progressive unpruning [22] limits the number of child nodes added to any node until a certain threshold is reached. The threshold can increase with the number of iterations, slowly upruning or adding more children to each node. This has been applied effectively to TUBSTAP [4] with an emphasis on attack actions. Progressive upruning can be paired with progressive bias [22] which uses heuristic knowledge to influence node selection. As the number of iterations increases, the heuristic influence decreases. Progressive bias could be an effective solution to TUBSTAP given the development of effective heuristic state evaluators such as phase division [23]. The phase division heuristic evaluator is based on unit positioning and matchups rather than simply HP and unit type such as in [4].

2.5 Hidden Information MCTS Related Works

Belief state MCTS (BSMCTS) [24] seeks to solve the non-locality problem of ISMCTS and PIMCTS. BSMCTS adds opponent guessing and opponent modeling to phantom games. In phantom games, players cannot access information the opponent has. Kriegspiel is a phantom game identical to chess, except each player's chess pieces are invisible to the opponent. If an opponent attempts to make an illegal move, they are prompted to make a different move. In the BSMCTS tree, each node contains a probability and a state. Each iteration of the algorithm creates a determinization like in ISMCTS but uses an opponent model to infer a determinization through historical moves. At player nodes, BSMCTS uses opponent guessing. Opponent guessing uses an online learning method to adjust the probabilities of states and select the next node using a utility function added to an exploration function similar to UCT. Opponent predicting occurs on opponent nodes, selecting the next state from a random distribution based on each state's utility and probability. Wang's BSMCTS outperformed in phantom Tic Tac Toe and phantom Go.

Whereas an opponent policy can be learned and help guide a player's determinizations through an opponent model, the player can also reveal his policy to the opponent. Cowling [25] addresses this issue of leaking information to the opponent through the concept of self-determinization in the game The Resistance. This game is based around hiding one's role while discovering which of the opponents are spies. Self-determinization introduces "impossible" worlds into the agent's determinization samples to hide its policy from inference. This led to bluffing behavior, a necessity in the Resistance.

Goodman [26] addresses another variant of information leakage that occurs in hidden information games in Re-determinizing ISMCTS (RISMCTS). Since ISMCTS creates determinizations at the root, both player and opponent nodes in the tree act with perfect knowledge of the player's true state. In reality, opponent nodes do not know the true state of the player's board. Thus, information about the player leaks down the tree causing ISMCTS to never consider positions where the opponent may act according to an imperfect determinization of the player. This is one reason why ISMCTS performs so poorly in the card game Hanabi. To solve this, RISMCTS redeterminizes the hidden information at each node so player and opponent nodes both act in accordance with hidden information.

Uriarte [27] defines the issue of Fake Omniscience in determinizations as when a player never tries to hide or gain information because they believe that they have perfect information. Solutions by Cowling and Goodman hide information both from the opponent and within the game tree, but information gain has largely been overlooked.

Additionally, ISMCTS, PIMCTS, and BSMCTS have been tested within TUBSTAP fog of war by [5]. The ISMCTS and PIMCTS algorithms worked as described in Section 2.3.1. Each determinization distributed enemies randomly among unseen tiles. In reality, this can be improved by tracking possible legal positions of enemy units from where they were last seen. PIMCTS outperformed ISMCTS and BSMCTS, however, the success of BSMCTS relies heavily on the opponent model and learning methods employed.

2.6 Summary

TUBSTAP with fog of war provides a flexible platform for testing multi-action hidden information games. Monte Carlo Tree search variants are an effective solution to searching complex game spaces and improve through various pruning techniques. Additionally, structuring the tree where each node is a single action prevents information loss that occurs when each node is a single turn.

Additional work has been done to prevent information leakage of one's policy to the opponent, prevent information leakage of one's knowledge into MCTS rollouts, addressing the problems of non-locality and strategy fusion in hidden information games, but there has yet to be research done on incentivizing information gain.

III. Methodology

To account for information gain in multi-action hidden information games, this chapter proposes the addition of an information gain incentive function. A risk function is also added to account for the inherent risk in taking information-gaining actions.

This chapter will first explain an accurate method to track possible enemy legal moves through the maintenance of a probability map. Then it explains the implementation of the information gain incentive and risk functions dependent on the probability map that is then integrated into PIMCTS and ISMCTS.

3.1 Maintaining a Probability Map

The following notation will be used in the implementation of information gain and risk algorithms. The map is represented by a set of tiles $t_{(x,y)} \in T \wedge t = \{plains, mountain, forest, road, ocean\}$. From the perspective of the active player, $f_i(l,t,h) \in F$ is a friendly unit, and $e_i(l,t,h) \in E$ is an enemy unit. l,t, and h correspond to the unit's location, type, and health respectively.

A visibility matrix $\{V_{(x,y)}\}$ tracks which tiles are visible. If $V_{(x,y)} = 1$, then $t_{(x,y)}$ is visible. If $V_{(x,y)} = 0$, then $t_{(x,y)}$ is not visible. V functions like a Boolean matrix but uses 1's and 0's for compatibility of operations with other matrices. V refers to the invisible tile where $V_{(x,y)} = 1$ when $t_{(x,y)}$ is not visible. The matrix $v(f_i)$ contains all tiles visible for unit f_i. The probability map $\{P_{(x,y,e)}\}(e \in E)$ tracks the probability of an enemy unit being on each tile. Each probability is between 0 and 1: $p_(x,y,e) \in P, 0 \leq p \leq 1$.

The probability map is initially a uniform distribution of possible positions that an enemy unit can be to guide determinizations. A simple way to determinize enemy units is by uniformly distributing them among all unseen tiles. This simple method allows for determinizations including illegal moves of enemies. The probability map prunes determinizations to exclude unrealizable states.

To help the probability map create useful information in the early turns of the game, the player assumes that enemy units begin in a mirrored configuration to their own. That is, for the first turn of the game, enemy units are distributed onto tiles opposite of friendly units with a probability of 1. A player determines the possible positions of enemy units in the probability map based on their total turn vision, the range of each enemy unit, and the probability map from the previous turn.

Figure 5 shows an example of how the probability map tracks the legal moves of enemies. The enemies in Map A are assumed to have been in squares e2 and e1 last turn. Each enemy has a movement of 2. Map A depicts the possible locations each enemy could be this turn (red squares for enemy 1, blue squares for enemy 2, and purple squares for both). Each friendly unit has a vision of 2. Map B shows the visibility of the player at the start of the turn. In map C, a unit is moved from a4 to a2 revealing more tiles. Note that tile c4 is out of range of both units, but since there was visibility of that tile at the start of the turn, it is included in the visible tiles. Before the next move, the player updates the probability map with the added information in Map D. Tile c2 cannot contain an enemy since it is now visible. Without any added belief, the probabilities of each enemy are uniformly distributed among the tiles. For determining the enemy range next turn, the probability map checks the range at each edge tile (tiles c1, d2, and e3 for enemy 2).

Psuedocode for updating the visibility matrix and probability map is displayed in Algorithm 4. To update ISMCTS and PIMCTS with the pmap information, call UpdatePmap at each iteration. Before the first iteration, the visibility map must be initialized with all elements to 0. At this time, the probability map is initialized with

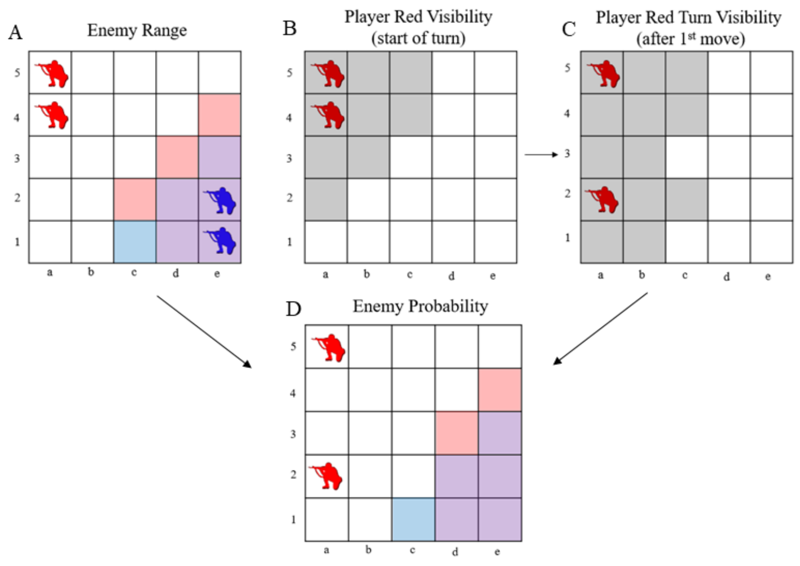
A
Enemy Range
B
Player Red Visibility (start of turn)
C
Player Red Turn Visibility (after 1st move)
D
Enemy Probability
a
b
c
d
e
1
2
3
4
5

Figure 5: Tracking possible enemy moves.

enemy units mirrored from friendly units starting positions with probabilities of 1.

3.1.1 Information Gain Incentive

The information gain incentive function combines information from the visibility matrix and probability map to create a heuristic for guiding MCTS selection. The damage calculation in Equation (1) largely favors the attacker over the defender. In TUSBTAP with hidden information, a unit cannot attack another unit that it cannot see. Finding enemy units is important for using attack advantage as well as for better organizing defense. If an enemy has a probability of 0 or 1 on a tile, then discovering that tile provides no information gain. A move generates a set of newly seen tiles G. A tile in the set of newly seen tiles $g \in G$ is considered significant if there is a probability of an enemy on that tile that is not 0 or 1. In Figure 5 D, tile c2 would be considered a significant tile. The sum of significant tiles can be calculated as:

$$\sum_{g_{x,y} \in G} g_{x,y}$$

```
Algorithm 4: UpdatePmap
Data: Initialized visibility map $V$, initialized probability map $P$, last unit
      that moved $f_{last}$
Result: Probability Map P that uniformly distributes enemy units among
        possible unseen tiles
1  updateVisibilityMap();
2  //update the pmap for enemies within seen tiles
3  foreach $e_i \in E$ do
4      foreach $v_{x,y} \in V$ do
5          if $v_{x,y} = 1$ then
6              if $e_i(l) = (x, y)$ then
7                  $P_{x,y,e_i} = 0$;
8              else
9                  $P_{x,y,e_i} = 1$;
10 //update the pmap for enemies not within seen tiles
11 if it is not the first turn of the game for player 0 and it is the first move this
     turn then
12     $reachable$ = matrix $\{x, y, e_i\}$ initialized to 0 ;
13     foreach $e_i \in E$ do
14         foreach $p_{x,y,e_i} \in P$ do
15             if $p_{x,y,e_i} > 0$ then
16                 $reachable_{x,y,e_i} = 1$;
17                 if $p_{x,y,e_i}$ is adjacent to a tile with $p_{e_i} = 0$ then
18                     $range$ = matrix of reachable tiles for enemy $e_i$ where
                         $range_{x,y}$ is 1 if tile $t_{x,y}$ is reachable, otherwise 0;
                         $reachable_{e_i} \cup range$;
19                 $reachable_{e_i} \cap \sim V$;
20     $P = reachable$;
21 Normalize $P$ so $0 < p_{x,y,e_i} < 1$
22 Function updateVisibilityMap():
23     if it is the first move this turn then
24         $V \leftarrow$ set all elements to 0;
25         foreach $f_i \in F$ do
26             $V \cup v(f_i)$;
27         else
28             $V \cup v(f_last)$;
29     return;
```

$$g(n) = \begin{cases} 1 & \text{if } 0 < p_{x,y,e} < 1 \\ 0 & \text{else} \end{cases}$$

Additionally, information gain becomes less important towards the end of the turn. If a player has four units, moving the first unit informs the next three moves this turn. However, moving the last unit does not inform any future moves this turn. Any gained information then becomes degraded upon the enemy's next turn. Thus, the last unit should receive less incentive for information gain. This is reflected in the incentive function below by dividing the number of active friendly units (units that can still be moved) by the total number of friendly units. Equation (3) calculates the number of significant tiles discovered for each enemy and divides by the total number of tiles with $p_{x,y,e} > 0$. This restricts the value between 0 and 1.

$$significant = \sum_{e_{x,y} \in E} \sum_{g_{x,y} \in G} g(n)$$
$$total = \sum_{p_{x,y,e} \in P} 1 \; if \; p_{x,y,e} > 0$$

$$ig_i = \frac{\mid F_A \mid}{\mid F \mid} \times \frac{significant}{total}; 0 \leq r_i \leq 1 \tag{3}$$

The information gain function value changes the value used during MCTS Selection as shown in Equation (4) with a constant β. Ideally, this modification should be applied at every node in Selection which requires maintenance of the pmap through each node of the tree. However, due to difficulties with this pmap maintenance throughout the tree, Selection uses Equation (4) only in the first layer of nodes in the tree. Any nodes after this layer use normal Selection with just UCT.

$$u(s) = (1 - \beta)\, UCT_i + \beta\, ig_s \tag{4}$$

After all simulations have been completed or time has run out, MCTS chooses a child node from the root with the highest win rate. Instead of using just win rate at this step, ISMCTS and PIMCTS with the information gain inventive function choose a child according to the win rate plus the information gain incentive.

3.1.2 Risk Implementation

Information gain is associated with risk. This risk is largely accounted for simply in the UCT function Equation (2) in ISMCTS and PIMCTS by conducting simulations on a node across different determinizations. To enhance the determinizations to better account for risk, a risk function applies a paranoid distribution of enemy units by increasing the chance of them being in unfavorable positions. It weights the probability map so that some determinizations are more likely than others.

Two units of the same type and hp fighting each other will far more often result in a win for the unit that attacks first. When moving into a position that reveals an enemy unit, that enemy unit will be able to attack with attack advantage next turn in most scenarios (since each unit has movement equal to or greater than their vision). Whereas identifying the unit allows a player to attack it, it also makes the identifier vulnerable to attack.

This risk modifies the probability map by applying a paranoid distribution of enemy units along the fringe. The fringe is considered the tiles adjacent to the outermost visible tile. Enemies that are placed on the fringe pose a risk to units that move closer to it. Additionally, fringe enemy units are more relevant to an existing fight as they are closer in range for a counterattack (clearer in the cases where movement is greater than vision). This risk function weights determinizations so that some are more likely than others.

Algorithm 5 applies this paranoid distribution by increasing the probability of a unit to be on a certain tile given its proximity to the fringe. Unseen tiles are given a higher value the closer they are to a seen tile. The squared value of the tile is multiplied by the probability of the unit being on the tile to increase the likelihood of placing an enemy near the fringe during a determinization. Figure 6 displays how the value of tiles is determined.

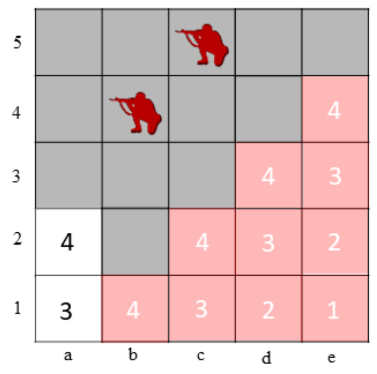

Figure 6: Applying uneven fringe weighting. Grey tiles represent turn vision. Red squares represent possible locations of enemy units. Numbers show proximity to seen tiles and the square of these values is used as weighting in the probability map.

Algorithm 5: ParanoidDistribution

```
    Data: Initialized visibility map $V$, Initialized probability map $P$
    Result: Probability map $P$ with increased weighting on fringes
1   initialize root as the current state of the game;
2   $c = 0$;
3   $w =$ set to a max value. Could use 2/3 map width;
4   $fringe = V$;
5   while $w > 0$ do
6       foreach $fr_{x,y} \in fringe$ do
7           if $fr_{x,y} > 0$ and $fr_{x,y}$ is adjacent to a tile with $fr = c$ then
8               $fr_{x,y} = w, c = w, w- = 1$;
9   Iterate over $x, y$ and multiply each element of $P_{x,y}$ by the squared value in
      $fr_{x,y}$; Normalize $P$ so $0 < p_{x,y,e_i} < 1$;
```

3.1.3 Summary

The information gain incentive and risk functions were created to jointly improve ISMCTS and PIMCTS in TUBSTAP. The incentive function increases the value of a node during selection based on the potential for gaining information. The risk function guides determinizations so that information-gaining units are susceptible to an enemy counterattack.

This chapter introduced a method for tracking the legal moves of the opponent using a probability map that guides determinations. This probability map can be modified to represent an opponent model and is done so using a paranoid distribution of enemy units. This is combined with an incentive function that is additive to the UCT evaluation during MCTS selection.

IV. Results and Analysis

To evaluate the effectiveness of the pmap and information gain and risk functions, this chapter reports the performance of each element tested separately and together with PIMCTS and ISMCTS on a variety of maps.

This chapter first presents three testing maps and defines five PIMCTS and ISMCTS variants. It explains the testing process and the chosen parameters for each variant. Next, it displays the results of the variants on each map before giving analysis of the performance to include an explanation for improving functionality.

4.1 Experiment Set Up

The information gain and risk functions were tested on three maps as shown in Figure 7. Map A provides a smaller game board where information is gained quickly, providing lower disambiguation. Map B provides a larger space with a more open center field. The scout has a large sight range making it valuable. It is strong against infantry and more resistant to tanks and artillery than infantry, making it relatively safe. Map C is an even larger map with 8 units and uses the fighter and attacker which each have a large sight range. They however are very vulnerable on this map given the large presence of anti-aircraft.

PIMCTS and ISMCTS with the information gain and risk functions will compete against each other to determine the effectiveness of information gain and risk functions separately and together. The MCTS variants are:

1. PIMCTS-Pmap/ ISMCTS-Pmap: PIMCTS and ISMCTS using the probability map described in Section 3.1 with a uniform distribution. The information gain and risk functions are not used.

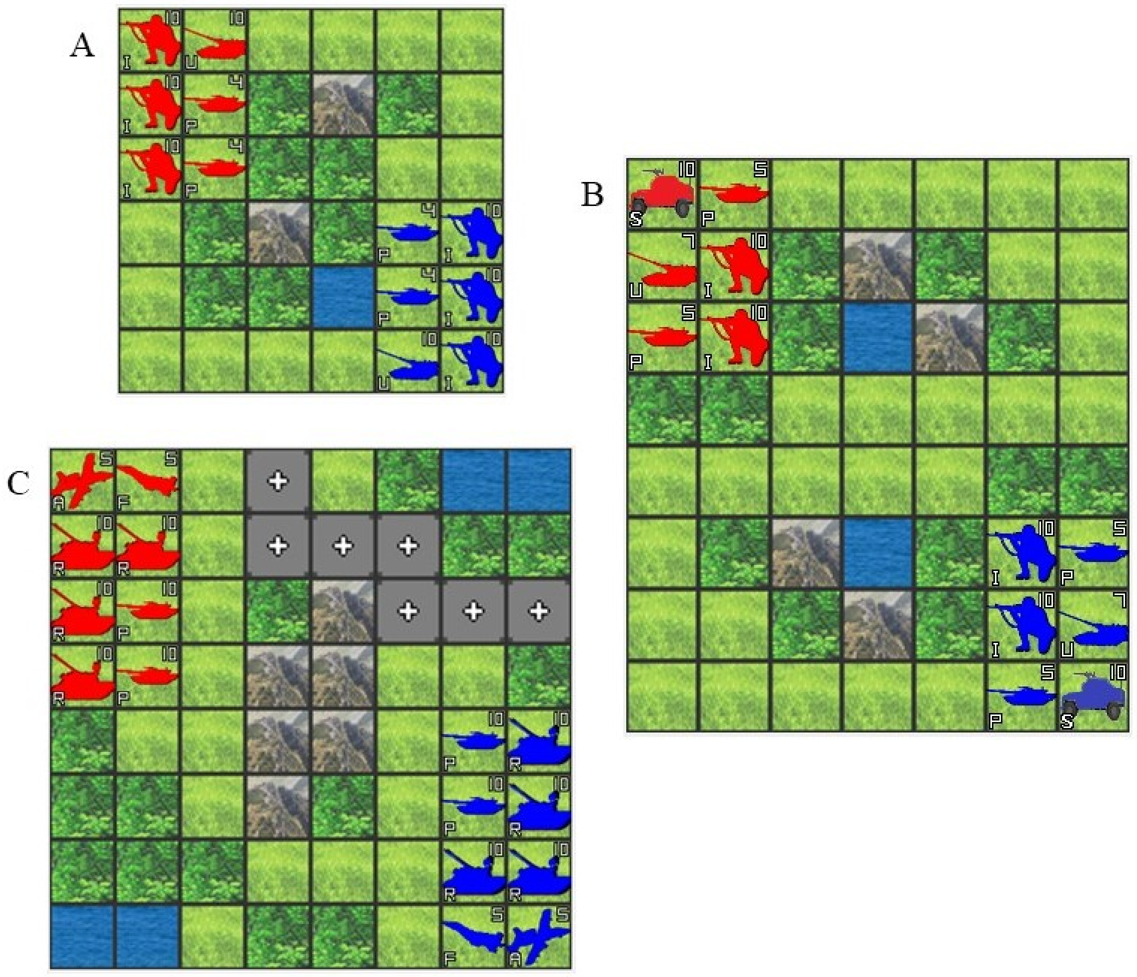

Figure 7: Test Maps A, B, and C.

2. PIMCTS-IG/ ISMCTS-IG: PIMCTS and ISMCTS using the information gain incentive function in Equation (4) and the pmap with uniform distribution.

3. PIMCTS-Risk/ ISMCTS-Risk: PIMCTS and ISMCTS using the fringe distribution applied to the pmap as depicted in Figure 6.

4. PIMCTS-IGR/ ISMCTS-IGR: PIMCTS and ISMCTS using the information gain incentive function and fringe distribution applies to the pmap.

5. PIMCTS-S/ ISMCTS-S: PIMCTS and ISMCTS without the use of a probability map. They instead use the simplified determinization method of distributing

enemy units uniformly among unseen tiles as used in [5].

Algorithms used a UCT constant α of 0.15 as used in [5] and a β of 0.05 if using the information gain incentive function. A parameter sweep on Map A using values between 0.05 and 0.30 determined β. A steep drop-off in performance occurred after a b of 0.2. Each algorithm used 10,000 iterations of MCTS. PIMCTS divided this into 20 determinizations of 500. Additionally, all MCTS rollouts choose attack actions most of the time when available.

For each map, PIMCTS-Pmap competed against each of its variants (PIMCTS-S, PIMCTS-IG, PIMCTS-Risk, and PIMCTS-IGR). The same was done for ISMCTS-Pmap and its variants. Next PIMCTS-Pmap, ISMCTS-Pmap, and cheating UCT competed in a round-robin tournament to determine the play strengths of PIMCTS and ISMCTS algorithms. Cheating UCT has complete knowledge of enemy unit locations but still can only attack enemies that are within its unit vision. Each matchup in the round-robin tournament consisted of 30 games and sides were switched between each game to account for turn 1 advantage or disadvantage.

4.2 Results

Each match receives 1 point for wins, -1 point for losses, and 0 points for ties. This generates a score between -30 and 30 which is then normalized between 0 and 1. Since each competes in 30 games, the normalized score is calculated as $(wins - losses + 30)/60$. A score with an even number of wins and losses results in a score of 0.5. Thus any score above 0.5 suggests better performance, scores below 0.5 suggest worse performance. A game with 18 wins, 9 loses, and 3 ties results in a score of 0.65.

Table 4 and Table 5 display results for testing PIMCTS and ISMCTS against their respective variants on Map A. Each variant competed against all other variants in 30 games. The score of each algorithm (player) is displayed in the rows of the tables;

each column is the opponent they played against. For example in Table 4, the score of PIMCTS-Pmap against PIMCTS-S is 0.72, demonstrating a higher win-rate and stronger performance of PIMCTS-Pmap.

Table 4: PIMCTS variants Map A

Opponent / Player	PIMCTS-S	PIMCTS-Pmap	PIMCTS-Risk	PIMCTS-IG	PIMCTS-IGR
PIMCTS-S	-	0.28	0.2	0.32	0.22
PIMCTS-Pmap	0.72	-	0.45	0.52	0.38
PIMCTS-Risk	0.8	0.55	-	0.57	0.48
PIMCTS-IG	0.68	0.48	0.43	-	0.48
PIMCTS-IGR	0.78	0.62	0.52	0.52	-

Table 5: ISMCTS variants Map A

Opponent / Player	ISMCTS-S	ISMCTS-Pmap	ISMCTS-Risk	ISMCTS-IG	ISMCTS-IGR
ISMCTS-S	-	0.27	0.28	0.33	0.3
ISMCTS-Pmap	0.73	-	0.58	0.43	0.38
ISMCTS-Risk	0.72	0.42	-	0.43	0.48
ISMCTS-IG	0.67	0.57	0.57	-	0.58
ISMCTS-IGR	0.7	0.62	0.52	0.42	-

PIMCTS-IGR/ISMCTS-IGR vs PIMCTS-Pmap/ISMCTS-Pmap reflects the added utility of the information gain incentive and risk functions when playing against an opponent that can accurately track opponent moves. For example PIMCTS-IGR had a score against PIMCTS-Pmap of 0.62, suggesting the information gain incentive with risk offset improved performance. Both ISMCTS and PIMCTS had an increase in performance with the risk and IG incentive functions (IGR) against the pmap variants on map A. IGR variants did not necessarily outperform risk and IG variants alone.

The addition of a pmap to accurately track opponent positions and prune illegal determinizations led to a much higher and consistent win rate over algorithms that used simplified determinizations (the S variant). Pmap, Risk, IG, and IGR variants all use the pmap and all perform much better than the S variant without the pmap.

Table 6 and Table 7 display results for testing PIMCTS and ISMCTS against their respective variants on Map B for another 30 games each.

Table 6: PIMCTS variants Map B

Player \ Opponent	PIMCTS-S	PIMCTS-Pmap	PIMCTS-Risk	PIMCTS-IG	PIMCTS-IGR
PIMCTS-S	-	0.28	0.37	0.32	0.4
PIMCTS-Pmap	0.72	-	0.53	0.53	0.35
PIMCTS-Risk	0.63	0.47	-	0.62	0.47
PIMCTS-IG	0.68	0.47	0.38	-	0.32
PIMCTS-IGR	0.6	0.65	0.53	0.68	-

Table 7: ISMCTS variants Map B

Player \ Opponent	ISMCTS-S	ISMCTS-Pmap	ISMCTS-Risk	ISMCTS-IG	ISMCTS-IGR
ISMCTS-S	-	0.37	0.35	0.33	0.3
ISMCTS-Pmap	0.63	-	0.47	0.7	0.57
ISMCTS-Risk	0.65	0.53	-	0.5	0.6
ISMCTS-IG	0.68	0.3	0.5	-	0.52
ISMCTS-IGR	0.7	0.43	0.4	0.48	-

On map B, PIMCTS-IGR again performed better than PIMCTS-pmap. However ISMCTS-IGR did not perform as well as ISMCTS-Pmap. ISMCTS-IG performed very poorly, but combination with the risk function (to create ISMCTS-IGR) did improve this poor performance. It is unclear why ISMCTS-IGR performed worse than the pmap variant on this map. The larger size of map B keeps the IG and risk functions more relevant throughout the game whereas, in map A, they are mostly used during the first few turns. Perhaps this more relevant information gain is more helpful to PIMCTS which can only sample over a small amount of determinizations. Any added knowledge then increases the accuracy of these determinizations. However, ISMCTS may not need the information gain incentive as much as the plentiful number of determinizations provide enough information. If this is the case, then perhaps the amount of information gain incentive should decrease over the course of the game. Or information gain incentive should be more selective. On map B, the increased performance from using a pmap is still evident as all variants outperform the S variant.

Table 8 and Table 9 display results for testing PIMCTS and ISMCTS against their respective variants on Map C for another 30 games each. Due to having to make corrections in the information gain function, this paper did not have enough time to retest information gain variants on Map C. Thus these data points are left blank in Table 8 and Table 9.

Table 8: PIMCTS variants Map C

Player \ Opponent	PIMCTS-S	PIMCTS-Pmap	PIMCTS-Risk	PIMCTS-IG	PIMCTS-IGR
PIMCTS-S	-	0.48	0.43		
PIMCTS-Pmap	0.52	-	0.53		
PIMCTS-Risk	0.57	0.47	-		
PIMCTS-IG					
PIMCTS-IGR					

Table 9: ISMCTS variants Map C

Player \ Opponent	ISMCTS-S	ISMCTS-Pmap	ISMCTS-Risk	ISMCTS-IG	ISMCTS-IGR
ISMCTS-S	-	0.37	0.55		
ISMCTS-Pmap	0.63	-	0.58		
ISMCTS-Risk	0.45	0.42	-		
ISMCTS-IG					
ISMCTS-IGR					

On Map C, the pmap still appears to outperform simple variants but less so. This is likely due to having a larger map with a smaller sight range. The fighter and attacker's additional sight range is mostly unused since they are completely vulnerable to anti-aircraft and placing them behind friendly units subtracts from their ability to gain vision. Over time, the unseen opponent's legal moves eventually become very similar to the unseen tiles by the player.

Tables 10-12 display the results of a round-robin tournament of PIMCTS-Pmap, ISMCTS-Pmap, and Cheating UCT on maps A, B, and C respectively. The pmap variants were chosen over the simplified variants given their consistent success. ISMCTS consistently outperformed PIMCTS, winning more often against both PIMCTS and Cheating UCT. The increased performance of ISMCTS is more evident on Maps B and C likely since these have more hidden information than Map A. ISMCTS avoids the issue of strategy fusion and samples across a much wider range of determinizations than PIMCTS leading to this increase in performance. Additionally, since ISMCTS uses one tree, it can search much deeper. Making more accurate determinizations through the use of the pmap likely allows ISMCTS to better leverage these advantages over PIMCTS.

Table 10: PIMCTS vs ISMCTS Map A

Opponent / Player	PIMCTS-Pmap	ISMCTS-Pmap	Cheating UCT
PIMCTS-Pmap	-	0.43	0.27
ISMCTS-Pmap	0.57	-	0.25
Cheating UCT	0.73	0.75	-

Table 11: PIMCTS vs ISMCTS Map B

Opponent / Player	PIMCTS-Pmap	ISMCTS-Pmap	Cheating UCT
PIMCTS-Pmap	-	0.32	0.18
ISMCTS-Pmap	0.68	-	0.38
Cheating UCT	0.82	0.62	-

Figures 9-11 display the average maximum search depths within one game of cheating UCT, PMCTS-S/IGR, and ISMCTS-s/IGR every 5 turns. Each of these algorithms competed against themselves in one game. This is only partially repre-

Table 12: PIMCTS vs ISMCTS Map C

Player \ Opponent	PIMCTS-Pmap	ISMCTS-Pmap	Cheating UCT
PIMCTS-Pmap	-	0.4	0.2
ISMCTS-Pmap	0.6	-	0.37
Cheating UCT	0.8	0.63	-

sentative of actual average search depths, since performance of an algorithm changes with its opponent. Lines that do not extend to the full turn count represent games that ended early. PIMCTS naturally has a lower depth since it must divide its computation budget across a number of trees. Cheating UCT decreases search depth overtime. This could be due to doing more exploration as it fails to find good moves later in the game, or by having an increased branching factor as pieces move toward the center of the board. The information gain incentive appears to lead to a deeper search for both ISMCTS and PIMCTS on maps A and B, but this does not hold on map C. More iterations are needed to determine the extent that depth is affected, but the graphs give a general overview of the tree structure.

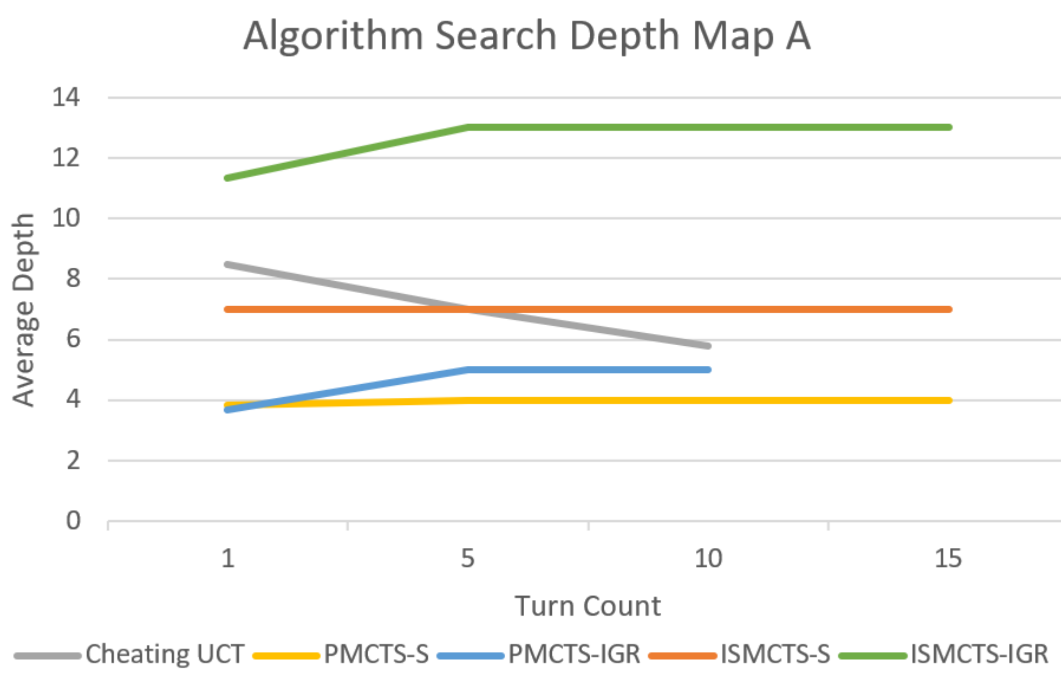

Figure 8: Average Search Depth Map A.

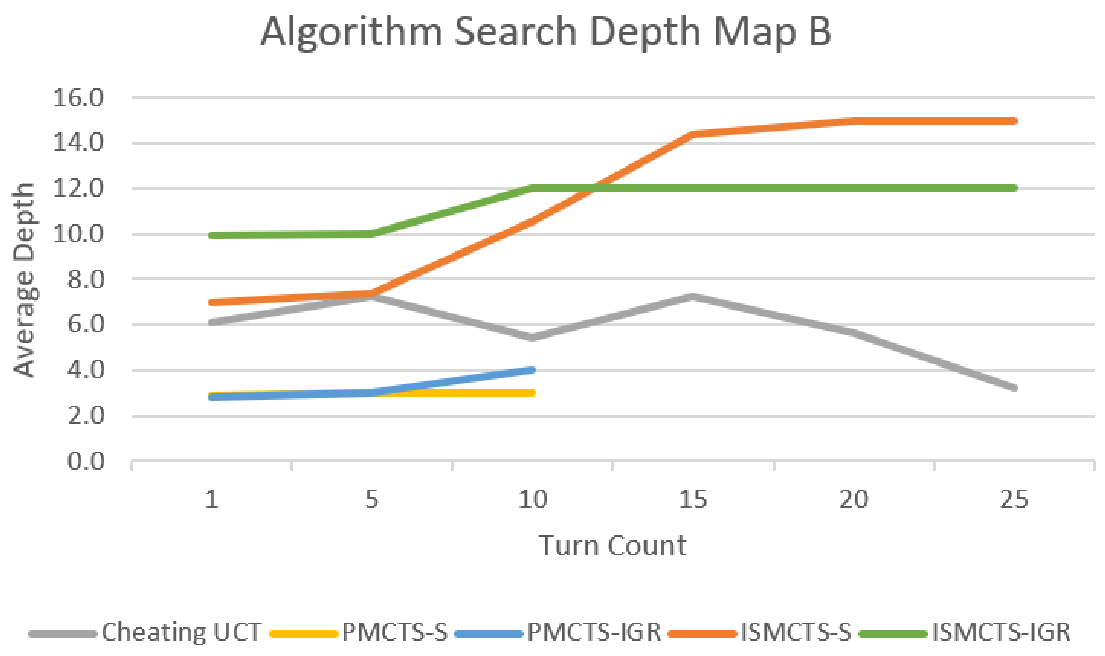

Figure 9: Average Search Depth Map B.

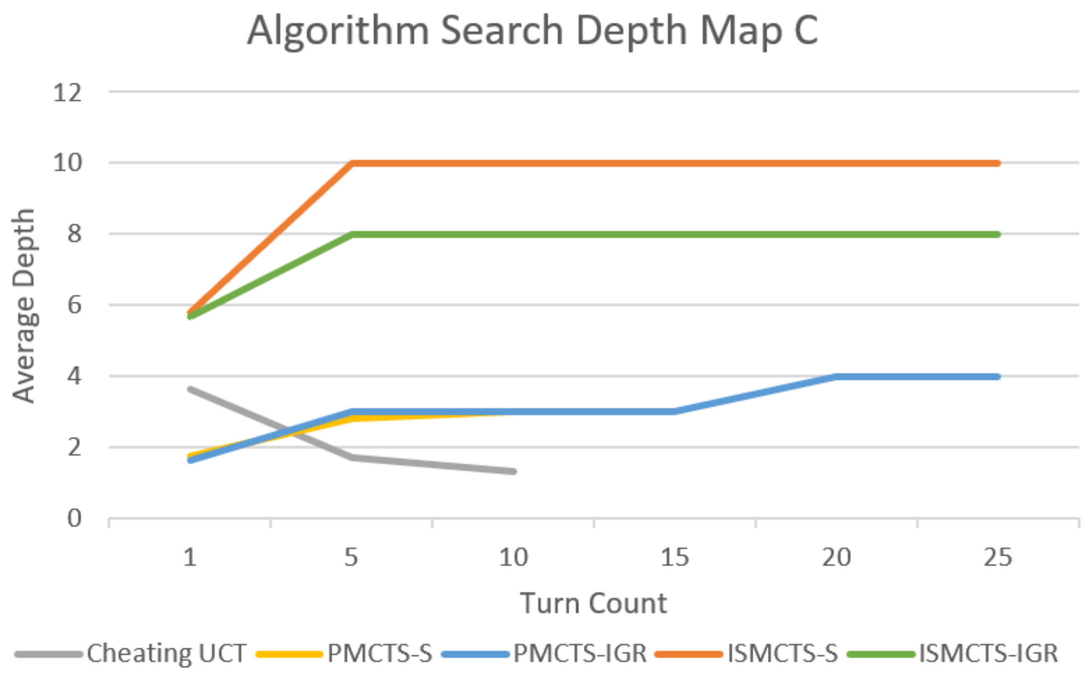

Figure 10: Average Search Depth Map C.

4.3 Analysis

Currently, the information gain incentive only is used in the Selection step at the first layer (a depth of 1) of the search tree. By applying this at only the first layer, MCTS still samples actions that provide information gain more frequently. However, since deeper nodes do not consider information gain, the player's policy is not fully reflected in the tree, causing inaccurate evaluation. This is flawed: an action candidate is evaluated based on the assumption that later actions (past a depth of 1) will not give information gain at all. To more accurately evaluate candidate actions, information gain should be applied to Selection at each node in the tree rather than just to the first layer.

Both the information gain incentive alone (IG) and risk (R) alone variants were more effective dependant on the map configuration and algorithm used (PIMCTS vs ISMCTS). Thus variants that include information gain incentive and risk functions

(IGR) saw increased performance when either of these functions improved performance. Due to the differing structures of PIMCTS and ISMCTS, information gain incentive and risk had differing effects. Information gain may be more important for PIMCTS since it is limited by the number of determinizations it can make (too many determinizations decreases individual tree depth for the same computational budget).

Adding the pmap to prune illegal moves from determinations proved to be very helpful in maps A and B, where a relatively small number of units could see about half of the map. In map C, where a large number of units still struggled to see half of the map, it was less effective. This is likely because as the game progresses without finding opponent units, the unseen opponent's legal moves eventually become equal to the unseen tiles by the player. In these situations, a more functional information gain incentive implementation could prove beneficial, to help overcome this pmap reduction. Additionally, MCTS may struggle to find any good moves at a certain level of complexity, making the advantages of the pmap less impactful.

Currently the incentive function only evaluates the score based on the revealed tiles that could contain an opponent, and how many friendly units have already moved. Many other factors may be considered when evaluating the value of information gain as well:

- Not all information gained is relevant to future actions. If a unit moves to uncover fog of war at a location out of range of active allies, then that information is irrelevant to future attacks this turn.
- An information-gaining action with some risk may give the opponent an equal amount of information for free, making it disadvantageous.
- An action may not gain information this turn, but provide more information on subsequent turns. Revealing tiles that have no chance to contain enemies

may not help with determining an action this turn, but may be useful when an enemy moves into that tile later.

- Information gain importance may change with game phases. In TUBSTAP, the initial turns are vital in organizing units to defend each other and command board presence while later turns focus entirely on exploiting enemy weaknesses. In this initial phase, information gain is much more important.

Likewise, the established risk determinization may be too simple to work in every scenario. Determinizations with the uneven fringe weighting on a large map may isolate enemy units from being within range of each other. There is more risk if there are multiple enemy units that will be able to counterattack than just one. Unit proximity to each other is important in defining a defensible position.

V. Conclusions

This report addresses the MCTS shortcoming of not considering information gain in hidden information games. This shortcoming becomes more predominate in hidden information games that also have multi-action elements as starting actions can inform later actions in the turn. The proposed risk and information gain incentive functions while demonstrating some increased performance, fail to do so consistently in ISMCTS and PIMCTS. Current implementation of the incentive function only applies the incentive during selection at the first layer of nodes. Using the information gain incentive function at each node in the selection steps of MCTS is necessary to ensure sampled action candidates have been evaluated according to a consistent selection policy. Additionally, the use of a more restrictive incentive function that identifies more relevant information may be more effective. When weighting determinizations with the risk function, other factors such as unit proximity should be taken into account. The implementation of a probability map that tracks enemy legal actions does improves performance over a simple distribution of enemies among unseen locations. Modifying MCTS hidden information variants to account for gaining information is still a necessary step in overcoming the limitations of determinizations. More work is needed to determine the advantage of effectively incentivizing information gain and offsetting inherent risk.

5.1 Future Work

Additional work for this report includes applying an information gain incentive at each node in the Selection step of MCTS rather than just the first layer of selection.

To do this, the current implementation needs to perform updates on a copy of the pmap throughout the game tree to properly calculate the information gain incentive at depths greater than one (at a depth of one, copying is not needed and selection can use the pmap at the root). Currently, the information gain incentive accounts for decay as a single turn progresses (providing less incentive for each friendly unit that has already moved). However, when calculating information gain incentive throughout the tree, this decay should occur at each depth, so that information gain is only considered at early nodes in the game tree. This implementation focuses on evaluating nodes more accurately according to the policy of information gain. Information gain in this implementation could also be used to effectively prune unpromising nodes and search deeper in the tree.

Exploring other dependencies on information gain in TUBSTAP such as player unit proximity to gained information would also be beneficial. A better utility function that only triggers that only incentivizes information-gaining actions under certain conditions could improve performance.

If found successful, information gain incentivization could be applied to BSMCTS or RISMCTS. Additionally, RISMCTS has not been tested in the TUBSTAP domain and may be effective to better reflect the opponent's policy. This would be done by using an opponent's pmap to create determinizations of the player at each opponent node.

Bibliography

1. Christopher J Keller, Richard Averna, and Donald Matcheck. How to master wargaming: Commander and staff guide to improving course of action analysis 20-06. *Mission Command Center of Excellence*, 2020.

2. Peter J Schwartz, Daniel V O'Neill, Meghan E Bentz, Adam Brown, Brian S Doyle, Olivia C Liepa, Robert Lawrence, and Richard D Hull. AI-enabled wargaming in the military decision making process. In *Artificial intelligence and machine learning for multi-domain operations applications II*, volume 11413, pages 118–134. SPIE, 2020.

3. Glennn Moy and Slava Shekh. The application of alphazero to wargaming. In *Australasian Joint Conference on Artificial Intelligence*, pages 3–14. Springer, 2019.

4. 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.

5. Connor M Pipan. Application of the monte-carlo tree search to multi-action turn-based games with hidden information. Master's thesis, Air Force Institute of Technology, 2021.

6. Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, 1(1):73–89, 2009.

7. Paul M Bethe. The state of automated bridge play. 2009. Unpublished.

8. Stuart Russell and Jason Wolfe. Efficient belief-state and-or search, with application to kriegspiel. In *IJCAI*, volume 19, page 278, 2005.

9. Jason Wolfe and Stuart Russell. Exploiting belief state structure in graph search. In *ICAPS Workshop on Planning in Games*, 2007.

10. 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.

11. Edward J Powley, Peter I Cowling, and Daniel Whitehouse. Information capture and reuse strategies in monte carlo tree search, with applications to games of hidden information. *Artificial Intelligence*, 217:92–116, 2014.

12. Rijul Nasa, Rishabh Didwania, Shubhranil Maji, and Vipul Kumar. Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game. *Int. Res. J. Eng. Technol*, pages 1637–1641, 2018.

13. Daniel Whitehouse, Edward J Powley, and Peter I Cowling. Determinization and information set monte carlo tree search for the card game dou di zhu. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 87–94. IEEE, 2011.

14. Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, 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):1–43, 2012.

15. Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

16. Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123, 1998.

17. Jeffrey Richard Long, Nathan R Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

18. Hendrik Baier and Peter I Cowling. Evolutionary mcts for multi-action adversarial games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

19. Nick Sephton, Peter I Cowling, Edward Powley, and Nicholas H Slaven. Heuristic move pruning in monte carlo tree search for the strategic card game lords of war. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–7. IEEE, 2014.

20. Bo Wu and Yanpeng Feng. Point-based incremental pruning for monte-carlo tree search. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 545–548. IEEE, 2017.

21. Naoyuki Sato, Tsubasa Fujiki, 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.

22. Guillaume M JB Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

23. Naoyuki Sato and Kokolo Ikeda. Phase division and simplification game offline tree search for phase evaluation value composition in turn-based strategy games. *Game Programming Workshop*, pages 61–68, 2015.

24. Jiao Wang, Tan Zhu, Hongye Li, Chu-Hsuan Hsueh, and I-Chen Wu. Belief-state monte-carlo tree search for phantom games. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 267–274. IEEE, 2015.

25. Peter I Cowling, Daniel Whitehouse, and Edward J Powley. Emergent bluffing and inference with monte carlo tree search. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 114–121. IEEE, 2015.

26. James Goodman. Re-determinizing mcts in hanabi. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019.

27. Alberto Uriarte and Santiago Ontanón. Single believe state generation for handling partial observability with mcts in starcraft. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 01–09–2022	**2. REPORT TYPE** Master's Capstone	**3. DATES COVERED** *(From — To)* Aug 2021 — Sept 2022

4. TITLE AND SUBTITLE Incentivizing Information Gain for MCTS in Hidden Information Game TUBSTAP	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Lervold, Nathan, 2nd Lt, USAF	**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-MAS-22-S-026
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RISB Dr. Braydon D. Hollis Email: brayden.hollis.1@us.af.mil Rome Research Site, Rome, NY	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	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

Hidden Information is a central mechanic in games like the Resistance or wargaming with fog of war, adding an extra layer of complexity for search algorithms. Monte Carlo Tree Search (MCTS) has gained much notoriety given its success in searching complex domains such as Go. Extensions to MCTS allow it to perform well in hidden information games such as Bridge, Kriegspiel (Chess), and Magic the Gathering. These MCTS extensions however fail to consider information gain, an important aspect of multi-action hidden information games as initial actions inform sequential decisions. This report proposes an information gain incentive function and a risk function to offset the risk of information gain. We implement the information gain incentive and risk functions into MCTS variants ISMCTS and PIMCTS which are then tested in the multi-action hidden information game TUBSTAP. Overall testing demonstrates promising results, but lack of consistency makes it largely inconclusive. Current implementation of the information gain incentive has flaws, and we offer a more effective approach.

15. SUBJECT TERMS

Tree search, Hidden Information Game, Multi-Action Game, Monte Carlo Tree Search (MCTS), Perfect Information MCTS (PIMCTS), Information Set MCTS (ISMCTS), Information Gain

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Gilbert Peterson, AFIT/ENG
U	U	U	UU	56	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-6565 x4281; Gilbert.Peterson@afit.edu