Improved Reinforcement Learning in Asymmetric Real-time Strategy Games via Strategy Diversity

Prithviraj Dasgupta and John Kliem
Distributed Intelligent Systems Section
Information Technology Division
U. S. Naval Research Laboratory, Washington, DC, USA
Email: {raj.dasgupta, john.kliem}@nrl.navy.mil

Abstract

We investigate the use of artificial intelligence (AI)-based techniques in learning to play a 2-player, real-time strategy (RTS) game called Hunting-of-the-Plark. The game is challenging to play for both humans and AI-based techniques because players cannot observe each other's moves while playing the game and one player is at a disadvantage due to the asymmetric nature of the game rules. We analyze the performance of different deep reinforcement learning algorithms to train software agents that can play the game. Existing reinforcement learning techniques for RTS games enable players to converge towards an equilibrium outcome of the game but usually do not facilitate further exploration of techniques to exploit and defeat the opponent. To address this shortcoming, we investigate techniques including self-play and strategy diversity that can be used by players to improve their performance beyond the equilibrium outcome. We observe that when players use self-play, their number of wins begins to cycle around an equilibrium value as each player quickly learns to outwit and defeat its opponent and vice-versa. Finally, we show that strategy diversity could be used as an effective means to alleviate the performance of the disadvantaged player caused by the asymmetric nature of the game.

Keywords: Real-time strategy game, asymmetric game, deep reinforcement learning, strategy diversity

1 Introduction

Real-time strategy (RTS) games have recently emerged as an attractive means of analyzing and evolving interactions between two or more players within a competitive setting. RTS games are characterized by several complex features including large action and state spaces, sparse rewards, and imperfect information in real-time about other players' moves. These features make it difficult for non-expert human players or simple computer algorithms to master RTS games. But at the same time, these features make RTS games an effective mechanism for understanding and responding in complex human-machine interactions such as cyber-security and military war-games [1, 2]. To address the complexity of playing RTS games, researchers have recently proposed deep reinforcement learning (RL) algorithms such as Alpha-Star with remarkable success in games such as StarCraft-II and Capture the Flag [3, 4]. However, the learning process or training used in these algorithms requires considerable time and computing resources to achieve expert level capability. Drawing inspiration from these results, in this paper, we investigate the suitability of different deep RL algorithms paired with a technique to diversify across a player's strategies and quickly improve its performance, in the context

of a 2-player anti-submarine warfare game called Hunting-of-the-Plark (HOTP). HOTP is an asymmetric, pursuer-evader game where the pursuer has superior capabilities in terms of its mobility and in perceiving the evader. This puts the evader at a distinct disadvantage even if it is able to predict the pursuer's movement strategy using deep RL. Most of the existing techniques on deep RL algorithms for playing RTS games are focused on outperforming the previously winning computer algorithm or champion human player, and win the game [4, 5]. A systematic comparison of the performances of different deep RL algorithms for learning to play a game is not the primary focus. Also, there has been limited focus on techniques that could break out of repeated stalemate outcomes in a game by disrupting the opponent, for example, when both players consistently play with equilibrium strategies. To address these issues, we make two main contributions in this paper. First, we analyze the performance of different deep RL algorithms including advantage actor critic learning (A2C), proximal policy optimization (PPO) and deep Q-networks (DQN) along with using techniques like self-play for modeling the opponent's moves and learning to play HOTP more effectively. We then investigate techniques to improve the disadvantage faced by the evader due to the asymmetry in the game using improved exploration of the game's problem space and strategy diversification approaches. We have evaluated our techniques within several rounds of game-play of HOTP within an AI gym environment. Our results show that the A2C algorithm achieves the best performance in terms of learning time and convergence of rewards for both players as compared to other deep RL algorithms. While self-play has been shown to be a suitable means to hone a player's abilities against its opponent, we show that if both players use self-play it results in each player repeatedly cycling around an equilibrium value around 50% wins. Finally, we show that strategy diversification by automatically switching between different learned strategies can be used by the disadvantaged player in the asymmetric game to break this cycle and improve its performance.

Our research results can be applied in several other domains that involve interactive decision-making between competing machine-machine or human-machine teams. For instance, one of the principal challenges in adversarial AI [6] is to guarantee that AI-enabled systems deployed in the real world can continue to operate robustly and reliably while interacting with other humans or machines around them that could potentially misguide the AI to produce incorrect or undesirable outputs. The self-play technique proposed in this paper could be used to address the adversarial AI problem by learning to identify malicious actions by agents or humans that are interacting with the AI and determining appropriate response strategies that ensure the AI's safe and risk-free operation. Closely related to adversarial AI, gamification techniques have been used very successfully to enable humans learn new strategies to solve a difficult problem. Examples include developing defenses against cyberattacks by playing an Escape Room game [7], solving medical diagnosis problems as a clinician game [8] and, developing problem solving skills by playing different roles in an ancient history-based game [9]. The strategy diversity technique proposed in this paper could be used in such scenarios to develop new strategies to defeat the opponent, for instance, by a cyber-defender to rapidly misdirect cyberattackers into wasting their attacks on decoys or low-value targets. Finally, war-gaming is a crucial exercise used for training military personnel [10]. Recent challenges in war-gaming environments such as the Alpha Dogfight trials [11] have demonstrated that deep RL-based techniques can outperform expert humans and sophisticated AI agents in complex tasks like air combat maneuvers. Our research advances this direction by developing deep RL techniques that respond strategically to aspects such as delayed or unavailable observations of the enemy, and asymmetric capabilities of the two sides. These results could be used to inform AI as well as humans to develop improved and appropriate responses in rapidly-evolving and complex war-gaming scenarios. In general, our research results are relevant to general machine learning scenarios where interactions with adversarial entities could be used to develop

new skills towards gaining a competitive edge and overcoming the adversary.

The rest of the paper is structured as follows: in the next section we discuss related work followed by a description of the main rules for playing HOTP. We then describe a formal representation of the game, the deep RL algorithms used to learn to play the game and the different movement patterns used by the deep RL algorithms on the evading player during learning. Sections 5 and 6 describe our experimental methods and evaluation of the RL algorithms and techniques to improve players' performances while playing the game, and finally we discuss future research directions and conclude.

2 Related Work

The main focus of research on agents that learn to play computer games including RTS games has been on how to determine winning strategies, that is, sequences of moves or actions that result in wins or higher rewards to the agent, by searching through the game-tree representation of a game. For larger games like Chess, Checkers and Go, and RTS games like StarCraft-II and Capture-The-Flag, it is infeasible to search through the entire game tree due to its large size. Probabilistic search methods like Monte Carlo Tree Search (MCTS) [12] have been the algorithm of choice for searching large game trees. Recently, MCTS has been extended with deep machine learning in algorithms such as AlphaGo and AlphaZero [13, 14]. Here, suitable moves provided by human experts or learned via self-play are used help guide the search towards winning outcomes. Results from these algorithms have shown that software agents using them can defeat human champion players in Chess, Go, Checkers and many other turn-based, strategic board games.

A more complex game-play setting is called partial observability where there is uncertainty in the information available to a player at each of its turns during the game, due to limited observability of the opponent's moves or noisy sensor data. This makes it difficult for a player to infer the current game state, or, in other words, which node in the game tree the game-play is currently at before making its move. Examples of partial information games include Reconnaissance Blind Chess, Poker, Hanabi and RTS games like StarCraft-II and Capture-the-Flag, and the game analyzed in this paper, HOTP. To handle partial observability, extensions of MCTS algorithms called Information Set MCTS [15], Belief state MCTS [16], Libratus [17] and zero shot coordination [18] have been proposed. In these algorithms, instead of calculating a single best response move to its opponent's move, a player calculates a set of best responses to each of the opponent's possible moves. The player then discards the opponent moves that are less likely to occur based on any partially observed information it might have obtained, for example, via using a 'probe' move to reveal the current state in a part of the game board. RTS games, specifically, add yet another level of complexity beyond partial observations, in that players no longer take alternating turns but can make moves in parallel or asynchronously while playing the game. Other challenges in RTS games include sparse rewards, delayed rewards and the credit assignment problem. To address these issues, researchers have proposed algorithms like AlphaStar [4] that use a technique called league play, where a player is matched up with opponents of increasing expertise levels while it is learning to play the game. Results from playing StarCraft-II using Alpha-Star showed that the agent could defeat human champions. In contrast to the games discussed above, a player in HOTP has no information or zero observability of its opponent's moves. Consequently, a player's information set cannot be pruned via inference like in partial observation games and players have to rely on histories of past game plays to learn from opponent's past behavior.

Recently, techniques to reduce the computational complexity of searching the game tree within the deep RL algorithm, and methods to make players' moves explainable to a hu-

man have also been researched [19–22]. These approaches could be used to complement the techniques in this paper to build faster and more human-interpretable techniques that can play HOTP, while addressing the player's limited observability of each others moves and the asymmetric nature in the game.

3 Hunting-of-the-Plark: Game Rules and Challenges

Hunting-of-the-Plark (HOTP) is an asymmetric, pursuit-evasion type, Naval warfare game that was first developed in 1987 under the title of Subsunk. The rules of the game were subsequently formalized in 2015 in [23]. Recently, a computer-based version of the game was developed by Montvieux Inc. along with a software interface for programming RL algorithms [24] to encourage the application of AI within military war-gaming problems.

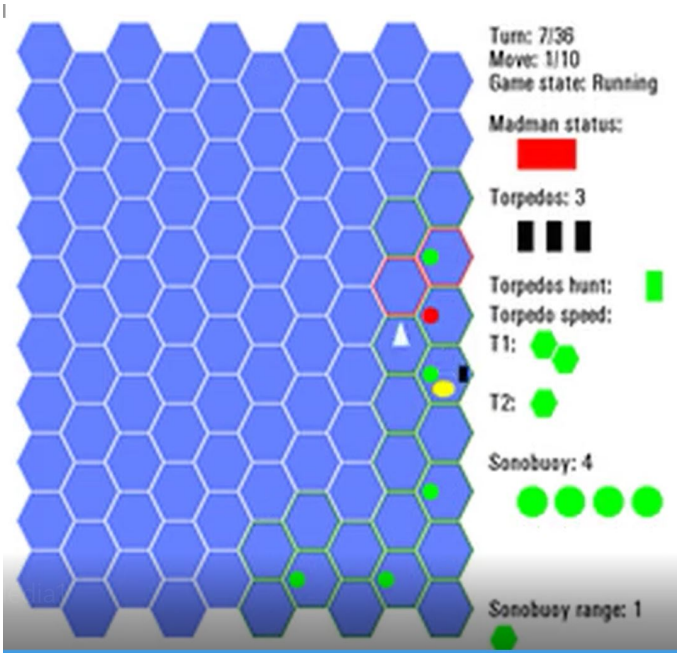


Figure 1: *Screenshot of the Hunting-of-the-Plark game interface towards the end of a game showing the Pelican (white triangle), Panther (yellow circle), sonobuoys (green circles) and torpedoes (black rectangles).*

HOTP is played on a rectangular game board that is divided into 900 hexagonal cells. The game has two players, a submarine called the Panther, that is the evader, and an aircraft called the Pelican, that is the pursuer. Panther's objective is to enter from the bottom of the board and exit from the top, while Pelican's objective is to detect and sink Panther using its on-board sensors, sonobuoys and torpedoes, respectively, before Panther exits the board. The number of each sensor that Pelican can carry is limited by a finite carrying capacity. Pelican and Panther take alternating turns and each gets 36 turns before the game ends. To play the game, Panther can start at any location that is within a 10-cell radius from the bottom-center of the board, and at each turn, it can move up to 2 cells from the location at the end of its last turn. Pelican can start at any location on the board, and at each of its turns, it can make up to 20 moves, one cell at a time and choose to deploy one of its available sensors at each cell visited during its movement. Sonobuoys do not move beyond their deployed location and can sense underwater movement (due to a torpedo or Panther) up to a range of 5 cells around them. Torpedoes can move 3 cells in the first turn and 2 cells in the second turn following their deployment in

the direction of their closest object (another torpedo or Panther). Pelican can become aware of Panther's presence if it visits (flies over) a cell currently occupied by Panther. However, Panther's presence information that is revealed to Pelican is partial and delayed - Pelican does not know the exact cell in which the fly-over happened and the information becomes available to Pelican only at the end of its turn. Consequently, Panther could move to a different cell during its turn before the Pelican gets its next turn and returns to one of its previously visited cells to deploy a torpedo. If a torpedo ends up in the same cell as the Panther, it either sinks, sustains damage or survives with one-third probability for each outcome; two damages sustained during the game sinks Panther. Panther wins if it is able to escape from the top of the board or if the Pelican exhausts all its torpedoes without sinking the Panther. The Pelican wins if it is able to sink the Panther before the end of the game. Any other outcome results in a draw. A detailed description of the game rules is available in [23].

Although the rules of the HOTP game appear straightforward, the interactions between the players can lead to complex scenarios. We identify three major factors contributing to HOTP's complexity:

- **Large decision spaces:** In HOTP, Pelican and Panther both have to make multiple, sequential moves at each of their turns - 20 moves for each Pelican's turn and 2 moves for each Panther's turn. Consequently, the decision space for selecting the best move from the set of available moves at each turn is a 20 dimensional space for Pelican and 2 dimensional space for Panther. The exponential size of the decision space makes the decision problem for selecting the best move infeasible to solve for either player using deterministic game search algorithms.

- **Limited information availability between players:** Pelican can only infer partial information about Panther's location if it is within range of a deployed sonobuoy while Panther gets no information about Pelican's movement strategy or deployed sensors at any point in the game. This inability to fully observe the opponent's moves leaves each player uncertain about the current state of the game. To ameliorate this uncertainty, each player needs to consider either every possible state of the game or the set of most likely states of the game calculated from the most recent information, if any, about the opponent's moves. Decision making under uncertainty introduces an additional level of complexity for the players to play the game effectively.

- **Asymmetric interactions among the players:** HOTP game rules make the game asymmetric with Panther at a disadvantage as it has less mobility compared to Pelican (2 moves versus 20 moves at each turn) and has no information about Pelican's visited locations, or sonobuoy and torpedo deployment locations. In contrast, to symmetric or zero-sum games, asymmetric games are more difficult to play because each player can no longer determine the opponent's strategy as the mirror image of its own strategy. Consequently, to respond effectively to opponent strategies, each player has to build a model of the opponent based off past interactions, use this model to determine opponent's likely strategies and then calculate appropriate responses to those strategies. Building precise opponent models from interaction histories introduces yet another level of complexity in playing asymmetric games.

To address the aforementioned complexities of HOTP, we make two contributions. First, we evaluate different deep RL algorithms to train agents that can autonomously play HOTP. Then, we investigate techniques that can be used by Panther to craft incrementally complex game-playing strategies so that it can overcome its disadvantage due to the inherent asymmetric nature of the game.

4 Learning to Play Hunting-of-the-Plark using Deep Reinforcement Learning

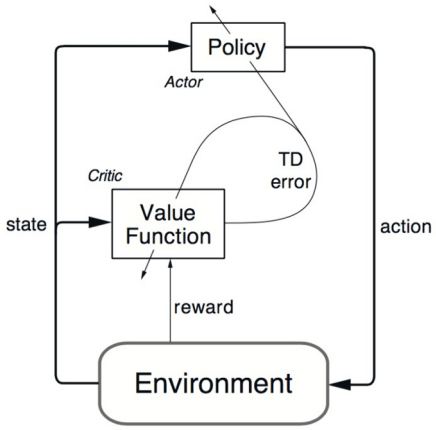


Figure 2: *Schematic of the Actor-Critic Algorithm [25]*

RL algorithms are a class of machine learning algorithms that use a reward based framework to enable an agent to learn how to perform sequential, decision-making tasks [26]. RL is formalized using a mathematical model called a Markov Decision Process (MDP). Formally, the MDP underlying RL is given by (S, A, T, R, γ) where S denotes the set of states, A denotes the set of actions for an agent, T denotes a state to action transition function specifying the forward dynamics model of the game where $T(s, a, s')$ is the probability of the agent reaching state s' when it takes action a at state s, $R : S \times A \rightarrow \Re$ denotes a reward function that gives a reward received by the agent by taking action a at state s, and γ is a discount factor. The objective of the agent is to determine a state to action mapping called a policy that maximizes the reward it receives while taking the actions towards performing the task. Mathematically, the policy $\pi : S \rightarrow P(A)$ is given by a state to action mapping that prescribes a probability distribution $P(A)$ over the action set. The objective of the RL algorithm is to determine an optimal policy that maximizes the expected rewards, that is, $\pi^* = \arg\max_\pi \mathbb{E}(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t))$.

RL algorithms can be broadly classified into two categories called value-based and policy-based. In the former, the agent learns an approximation of the expected future rewards from each state in its state space, called the value function, while in the latter, it directly approximates the policy function using a gradient-based technique. Deep RL algorithms implement the RL algorithms' policy using a deep neural network called a policy network, that is parameterized using a set of network edge weights denoted by θ. The objective of deep RL algorithms is to determine the value of parameter θ so that the policy realized via the policy network maximizes the reward the agent receives while performing its task. The reader is referred to [26] and [27] for excellent overviews of RL and deep RL algorithms respectively.

We have used three recent deep RL algorithms - one value function approximation based algorithm, Deep Q-networks (DQN) [28], and two policy gradient-based algorithms, Proximal Policy Optimization (PPO) [29] and Advantage Actor-Critic Learning (A2C) [30], for learning to play the HOTP game. DQN is a value-based RL algorithm that tries to estimate the action-value function by using a deep neural network. Formally, the action-value function is specified as a table of state-action pairs called the Q-table, denoted by $Q(s, a)$. Each state-action pair entry in the Q-table is updated using the equation: $Q(s_i, a_i) \leftarrow Q(s_i, a_i) +$

$\alpha(R_i + \gamma max'_a Q(s_{i+1}, a') - Q(s_i, a_i)$, where α is a learning rate parameter. The Q-table entries are updated until convergence, and, at that point, they give the optimal Q-values, $Q^*(s, a)$. When the agent performs its task, it selects actions corresponding to optimal Q-values, which guarantees that it maximizes its expected rewards. In DQN, because the number of state-action pairs can be very large, instead of representing the Q-values as a table, they are estimated using a function approximator such as a neural network with parameter θ. The Q-function is then denoted as $Q(s, a, \theta)$ and the objective is to determine the optimal value for parameter θ, θ^*, such that $Q(s, a, \theta^*) \approx Q^*(s, a)$. For determining θ^*, the loss for the i-th update of the Q-function is given by $L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(y_i - Q(s, a; \theta_i))^2]$ where $y_i = \mathbb{E}_{a' \sim \pi}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | S_t = s, A_t = a]$ and the expectation is taken over transitions of the form state, action, reward, next state tuples denoted in the equation by (s, a, r, s'). The loss is minimized using a stochastic gradient descent approach. Another extension in the DQN algorithm over the conventional Q-learning algorithm is an experience replay buffer where transitions made during learning are added to the replay buffer. The loss and its gradient are then computed using a mini-batch of transitions sampled from the replay buffer. Further details of the DQN algorithm are in [28].

PPO and A2C fall under the class of policy gradient-based RL algorithms. In these algorithms, the policy to be learned is again parameterized by θ. The general idea in policy gradient RL algorithms is to repeatedly perform three steps until convergence: a sampling step where multiple trajectories are sampled using the current policy, a policy evaluation step where a performance measure for the current policy is calculated from the sampled trajectories, and a policy update step where the policy parameter θ is updated using the performance measure calculated in the policy evaluation step. Mathematically, let $\tau_i = (s_{i,t}, a_{i,t} \ : \ t = 0...|\tau_i|$ denote the i-th sampled trajectory, and, N, the number of sampled trajectories. For the policy evaluation step, the performance measure of the policy as a function of current value of parameter θ is given as $J(\theta) = \mathbb{E}_{\tau \sim p_{\theta(\tau)}} \sum_t R(s_t, a_t)] \approx \frac{1}{N} \sum_{i=1}^{N} \sum_t R(s_{i,t}, a_{i,t})$. The policy update step is given by $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$, where α is the learning rate parameter and $\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\sum_{t=1}^{T} \nabla_\theta log \pi_\theta(a_{i,t}|s_{i,t}))(\sum_{t=1}^{T} R(s_{i,t}, a_{i,t})$.

The actor-critic framework extends the policy gradient RL approach by separating the policy evaluation and policy update steps. The critic component is responsible for the policy evaluation step and updates the Q-value function, denoted by $\hat{Q}_\omega(s, a)$, where ω is a Q-function parameter. The actor component takes the updated Q-value provided by the critic and does the policy update step using gradient descent: $\Delta\theta = \alpha \nabla_\theta (log_{\pi_\theta}(s, a)) \hat{Q}_\omega(s, a)$. In the next step, the critic uses the updated policy determined by the actor for its value update. The combination of value- and policy updates enables the actor-critic algorithm to learn more efficiently than using a value-based or policy-based approach separately. Figure 2 illustrates the interaction between the critic's value update and actor's policy update steps. A variant of the actor-critic algorithm that we have used here, advantage actor-critic (A2C), substitutes the Q-function with an advantage function, $A_\omega(s_i, a_i) = R(s_i, a_i) + \gamma Q_o mega(s_{i+1}, a_{i+1}) - Q_\omega(s_i, a_i)$, inside the actor's policy update step. This results in favoring policy updates towards higher reward actions. The interested reader is referred to [27] for further details of the A2C algorithm.

Finally, the proximal policy optimization (PPO) algorithm resolves a deficiency of the gradient descent in the policy update step of the A2C algorithm by limiting the step size of the loss function of θ using a clipped function, $L^{CLIP}(\theta_i) = \hat{\mathbb{E}}_i[\min(r_i(\theta_i)\hat{A}_i, clip(r_i(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_i]$, where $r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}$ gives the ratio between the action probabilities after and before the policy update, $\hat{A}_i$ is the advantage function and ϵ is a hyper-parameter. This prevents the algorithm from making too small updates to θ and converging too slowly, or, making very large updates to θ and overshooting θ's optimal value. Further details of the PPO algorithm are in [29].

4.1 HOTP Game State and Action Spaces, Reward Function

For use with a deep RL algorithm, the HOTP game has been modeled in the form of a state space comprising possible states in the game and a corresponding action space, comprising actions that Pelican and Panther could take at each state. The different attributes or features of the HOTP state space that are available to the Pelican and Panther are given in Table 1. In the HOTP game implementation, Panther is given access to the Pelican features marked with an asterisk, but these features are not included in Panther's reward function. So, although the Panther has this information, it does not utilize the information in its learning and behaves agnostic to this information. We use S_{pel} and S_{pan} to denote the features that are available to Pelican and Panther respectively.

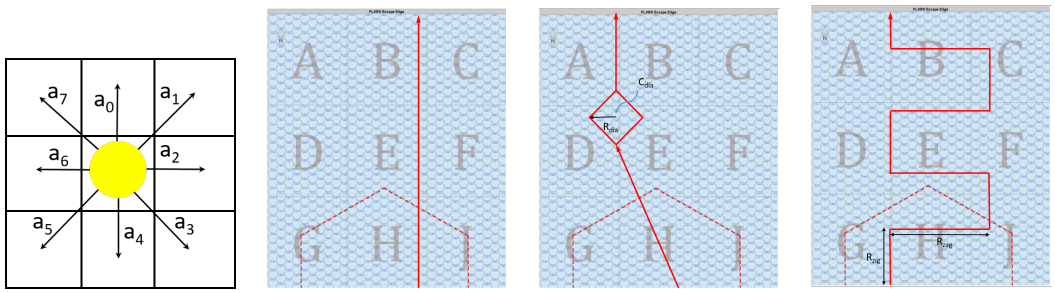
Table 1: *Features of the HOTP game state space that are available to Pelican and Panther respectively. *: feature available but not used for learning.*

State Space Feature	Notation	Pelican	Panther
Board size (width X depth) (measured in hexes)	$W \times D$	Yes	Yes
Pelican location	x_{pel}, y_{pel}	Yes	No*
Madman status	$mm \in \{0, 1\}$	Yes	No*
No. of Pelican moves remaining per turn	M_{pel}	Yes	No
Max. sonobuoy range	r_{sb}	Yes	No*
No. of sonobuoys remaining	n_{sb}	Yes	No*
Sonobuoy location	x_{sb}, y_{sb}	Yes	No*
Sonobuoy On/Off	$on_{sb} \in \{0, 1\}$	Yes	No
No. of torpedoes remaining	n_{tor}	Yes	No*
Torpedo speeds	$v_{tor} \in \{0, 1, 2, 3\}$	Yes	No*
Torpedo location	x_{tor}, y_{tor}	Yes	No*
No. of Panther moves remaining per turn	M_{pan}	No	Yes
Panther location	x_{pan}, y_{pan}	No	Yes

Each player, Pelican and Panther, has 8 possible actions at each state that it can take to move to one of the 8-neighbor cells from its current location. We denote the actions a_0 through a_7 in a clockwise manner starting the cell directly north of the player's current location, as shown in Figure 3(a). If an action takes either player to a cell outside the game board limits, the action is flagged as illegal and the player remains at its previous cell.

4.2 Overcoming Game Asymmetry

HOTP game rules make the game asymmetric as Pelican and Panther have disparate capabilities and information about each other during the game. As shown in Table 1, Panther is at a disadvantage compared to Pelican because it has access to less information about the game state. For example, before taking each of its turns, Pelican could infer Panther's approximate location if Panther is within range of any of its deployed sonobuoys, and determine possible locations to move through and deploy sonobuoys and torpedoes. Panther, on the other hand, is not able to get any information about the location of Pelican or its deployed sonobuoys, and cannot determine potentially safe locations to move to. Pelican can also move 10 times faster



(a) (b) (c) (d)

Figure 3: *(a) Possible actions of Panther, and, different movement patterns used by Panther: (b) Move North, (c) Diamond, (d) Zig-Zag on the Hunting-of-the-Plark game board. The area between the red dashed line area and the bottom of the board are the locations from which Panther could start its movement.*

than Panther (upto 10 Pelican moves for 1 move of Panther in each turn). We investigate two methods to overcome this disadvantage, for Panther, as mentioned below:

- *Environment Exploration.* Explore the environment during learning more effectively to infer possible placement locations of Pelican's sonobuoys and avoid them, and,
- *Strategy Diversity.* Prevent Pelican from discerning Panther's movement strategy by diversifying across different Panther movement strategies, so that Pelican cannot place sonobuoys on Panther's route to detect it.

To achieve Panther's exploration objective, we considered three deterministic or guided movement patterns with increasing spatial distribution or coverage. The different Panther movement patterns are illustrated in Figure 3(b)-(d) and described below:

- *Non-guided (random)* (`NG`). (not shown in Figure 3) In the non-guided pattern, Panther selects a random direction from its current location to move to. It is used as a baseline to compare the effect of the other, guided exploration patterns.
- *Move North* (`MN`). This is the basic, naive movement pattern provided with the HOTP game implementation, where at each move Panther moves to the cell that is directly north of its current location.
- *Diamond* (`DI`). The diamond movement pattern adds non-determinism to the Move North pattern. It is executed by selecting a diamond-center location C_{dia} at random on the board and executing a spiral pattern (approximated as a diamond shape) with radius R_{dia} centered at C_{dia}. Panther moves in a Move-North pattern to reach the perimeter of the diamond pattern from its start location at the bottom of the board, as well as to continue towards the top of the board after finishing the diamond pattern.
- *Zig-Zag* (`ZZ`). In the zig-zag pattern, Panther moves upwards using a zig-zag or ladder-like pattern, with R_{zig} and R_{zag} denoting the height and width of each leg of the ladder respectively.

Note that although these movement patterns are used by the Panther only during the exploration phase of its RL algorithm, they end up influencing both Panther's final learned movement strategy to evade Pelican as well as Pelican's learned sonobuoy and torpedo placement strategy to capture Panther.

For strategy diversity, we investigated a mechanism where Panther switches between the different strategies it has learned until it finds a strategy that performs best in terms of wins over a certain time window (measured in number of games). The Panther continues to use this best performing strategy until its number of wins over the time window falls below the second best strategy's performance, possibly due to the Pelican adapting its strategy. When this happens, Panther starts another strategy switching cycle to determine the new best strategy against Pelican.

Algorithm 1: Training and testing procedure for Pelican and Panther agents in HOTP game experiments.

input : RL_{alg}: RL algorithm to use for training agents
ag_{pel}, ag_{pan}: Pelican and Panther agents (post warm-up)
t_{train}: No. of training iterations
n_{bat}: No. of batches to split training into
n_{test}: No. of games to test trained agents with
`is_SelfPlay`: $\{True, False\}$

output: ag_{pel}, ag_{pan}: Trained Pelican and Panther agents

Procedure `train-and-eval-agents` $(RL_{alg}, ag_{pel}, ag_{pan}, t_{train}, n_{batch}, n_{test})$
  $ag^0_{pel}, ag^0_{pan} \leftarrow ag_{pel}, ag_{pan}$
  $t_{bat} \leftarrow t_{train}/n_{bat}$
  for $i = 1$ *to* n_{bat} **do**
    if *not* `is_SelfPlay` **then**
      $(ag_{pel,opp}, ag_{pan,opp}) \leftarrow (ag^0_{pan}, ag^0_{pel})$
    end
    else
      $(ag_{pel,opp}, ag_{pan,opp}) \leftarrow (ag^{i-1}_{pan}, ag^{i-1}_{pel})$
    end
    for t_{bat} *iterations* **do**
      $ag^i_{pel} \leftarrow$Train ag^{i-1}_{pel} playing against $ag_{pan,opp}$ using RL_{alg}
    end
    for t_{bat} *iterations* **do**
      $ag^i_{pan} \leftarrow$Train ag^{i-1}_{pan} playing against $ag_{pel,opp}$ using RL_{alg}
    end
    for n_{test} *iterations* **do**
      Record outcome - win, loss, draw, rewards of ag^i_{pel} and ag^i_{pan} playing against each other
    end
  end

5 Methods

We have addressed three main research questions to validate the performance of our proposed techniques:

1. How do the three deep RL algorithms, DQN, PPO and A2C, compare in terms of convergence of rewards as well as the magnitude of rewards for both Pelican and Panther?

2. Do the different movement strategies - `Move North (MN)`, `Zig-Zag (ZZ`, and `Diamond(DI)`, used by Panther to explore the environment during training, influence the rate at which it learns to respond and win against the Pelican?

3. Does strategy diversity enable Panther to improve its performance if it is faced with a stronger Pelican and overcome its disadvantage due to the asymmetric nature of the game rules?

Materials and Equipment. All our experiments were performed within the AI gym environments for the HOTP game available at [24]. The experiments were run on a server with 200GB RAM, four 2.7GHz 12-core Intel Xeon CPUs each with 19 MB cache, two Quadro P5000 GPU each with 16 GB RAM, with Ubuntu 20.10 as the operating system. For evaluating the performance of the different deep RL algorithms and different Panther movement strategies, we performed a series of experiments while varying the RL algorithm and the settings of the game. Each experiment consisted of three phases, as mentioned below:

- **Warm-up.** During this phase Pelican and Panther start with an untrained policy network (with random weights) and play 30 games to train their respective neural networks.

- **Training.** In this phase, each agent starts with the trained model after the warm-up phase and further trains the policy network against its opponent using the `train-and-eval-agents` method described in Algorithm 1. The training procedure is divided into batches. At the start of each batch, each agent selects the opponent to train against (lines $5-10$). If the agents are not using self-play, the opponent is the initial, post-warm opponent agent. If using self-play, the opponent is the trained opponent model from the previous training batch. The training is done using the deep RL algorithm being evaluated and the opponent agent's model is kept fixed during the agent's training (lines $11-18$). Each iteration consists of 30 games between the agents.

- **Evaluation.** At the end of each batch, the performance of the trained models of each agents are evaluated by playing n_{test} games against each other (lines $17-19$). To even out first mover advantages during the games, Pelican is the first mover in half of the games, while Panther is for the remaining half.

Table 2: *Rewards for different Pelican and Panther actions and game outcomes in* `plark-env-v0` *environment.*

Player	Action	Reward
Pelican	Drop sonobuoy, low spacing (sonobuoy ranges overlapping)	-0.2
Pelican	Drop sonobuoy, high spacing (sonobuoy ranges not overlapping)	$+0.5$
Either	Illegal Move	-0.1
Either	Win Game	$+1$
Either	Lose Game	-1

Game Settings and Rewards. HOTP can be played on three different board sizes or maps: small (10×10), medium (10×10) and large (30×30) hexes. We used the gym environment called `plark-env-v0` for our experiments. In this environment, Pelican has up to 10 moves at each turn in the small and medium maps, and up to 20 moves for each turn in large map, while the Panther has 1 move each turn in all (small, medium and large) maps. For most of our

Table 3: *Pelican and Panther Variables Names and Possible Values.*

Pelican Variables	
Training Type	`NO`: No learning `RL`: RL only (no self-play) `SP`: Self-play w/ RL
Gaming Resources	`n`: No. of sonobuoys available to drop
Panther Variables	
Training Type	`NO`: No learning `RL`: RL only (no self-play) `SP`: Self-play w/ RL
Start location	`FS`: Fixed start loc `RS`: Random start loc
Movement Pattern (during exploration phase of RL)	`NG`: Not-Guided `MN`: Move North `ZZ`: Zig-Zag `DI`: Diamond
Trained Against	`(NO/RL/SP)n`: Trained against `Pel-(NO/RL/SP)-n`

experiments, we used the small map. The action-reward structure for the `plark-env-v0` environment is given in Table 2.

Agent Naming Convention. Pelican and Panther agents can have numerous configurations depending on the different variables that characterize their behavior and training procedure. To concisely and legibly identify the configuration of a Pelican or Panther agent, we have used an agent naming convention based on the variables that determine the agent's behavior, as given in Table 3. Each agent is identified with a string given by `<agent-prefix>-<var1>-<var2>-...`. For example, for Pelican agent, `Pel-NO-5` denotes a Pelican agent that does not use any learning to determine sonobuoy placement locations and drops 5 sonobuoys at fixed, pre-determined locations, `Pel-SP-3` denotes a Pelican agent that drops 3 sonobuoys at locations that are learned using RL with self-play, and `Pan-RL-FS-ZZ-SP3` denotes a Panther agent that uses RL for training, starts from a fixed location at every game with Zig-Zag as its exploration method during RL training, and trains against a `Pel-SP-3` Pelican. There are additional variables for the agent configurations depending on the deep RL algorithm used and its hyper-parameters, parameters used during training the RL algorithm, etc. For the sake of legibility of agent names, we have specified these additional variables alongside the respective experiments.

6 Experimental Results

In this section, we discuss the results of the experiments we performed to validate the aforementioned research questions:

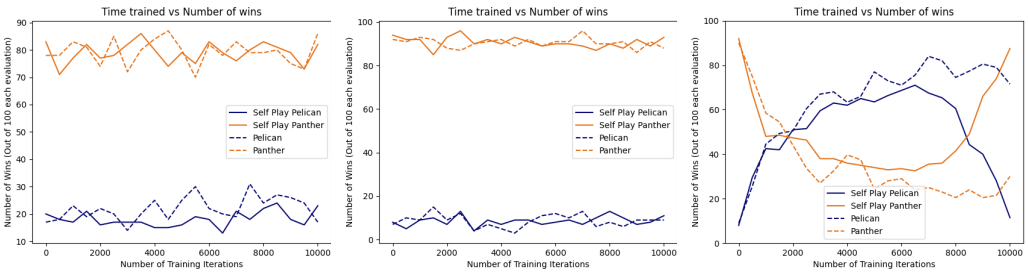
6.1 Performance Comparison of Different Deep RL Algorithms

For our first experiment, we evaluated the performance of three deep RL algorithms, DQN, PPO and A2C, for learning to play HOTP. The training and testing procedure used for this experiment is shown in Algorithm 1. The different inputs of the algorithm used for these experiments were $RL_{algo} = \{DQN, PPO, A2C\}$, $ag^0_{pel} =$ `Pel-NO-5`, $ag^0_{pan} =$ `Pan-NO-RS-ZZ-NO5`, $t_{train} = 10^4$, $n_{bat} = 4$, $n_{test} = 100$. The different hyper-parameters

used in our experiments for the different RL algorithms are given in Table 4. Most of the hyper-parameters are set of default values given in stable-baseline2. For some of the hyper-parameters, we used a hill-climbing technique to determine suitable values that gave better performance. The results of the experiments without and with self-play for both Pelican and Panther and are shown in Figure 4. Our main findings were that DQN and PPO failed to converge even after 10^5 training iterations and Pelican could never learn to effectively place sonobuoys to detect the Panther. A2C was the most effective RL algorithm as both Pelican and Panther learn to play the game after the first batch of training iterations. These results indicate the purely value-based or policy gradient-based RL algorithms like DQN and PPO are not suitable for learning to play HOTP game quickly, especially for the Pelican, as the large state and action spaces coupled with sparse rewards for most actions end up requiring substantial times for these algorithms to explore the state-action space effectively for determining a suitable policy. In contrast, the interspersing of policy evaluation update and policy update by the critic and actor respectively in A2C first enables the Pelican to learn a policy to capture the Panther, and subsequently enables both agents to learn policies to defeat each other.

Table 4: *Hyper-parameters of the different RL algorithms used in our experiments.*

Parameter	DQN	PPO	A2C
Gamma	0.99	0.99	0.99
Number of Steps	N/A	128	5
Value Function Coefficient	N/A	0.5	0.25
Entropy Coefficient	N/A	0.01	0.01
Learning Rate	5×10^{-4}	2.5×10^{-4}	1×10^{-4}
Exploration Fraction	0.1	N/A	N/A
Prio. replay	False	N/A	N/A



(a) (b) (c)

Figure 4: *Comparison of different deep RL algorithms, (a) DQN, (b) PPO, (c) A2C, for learning to play the HOTP game by Panther and Pelican agents. Solid (dashed) lines show scores for agents using (not using) self-play during training.*

6.2 *Performance Comparison of Different Pelican versus Panther Match-ups*

For our next set of experiments, we evaluated the effect of different Pelican and Panther encounters. For all our experiments under this category, we used A2C as the RL algorithm for both agents, training was done over $10,000$ iterations. We investigate two research sub-questions under this category:

1. **Learning Improvement Via Exploration.** Under what conditions is exploration of the environment effective in improving Panther's performance towards overcoming the game asymmetry?

2. **Learning Transferability.** Can a Panther perform well against a weaker or stronger Pelican than the one it trained against?

Table 5 shows the results for pairings between different training regimes of Panther and Pelican agents. The values in each cell denote the number of wins out of 100 games received by the Panther in that cell's row while playing against the Pelican in the corresponding column. Because 50 out of 100 wins is the break even point for each player, we consider Panther to have overcome the asymmetry in the game if it gets more than 45 wins. For representing the table data compactly, we have used the agent naming convention in Table 3 with the sub-columns inside each column representing more aggressive explorations by Panther going from left to right. So, `FS-MN` represents a Panther that uses fixed start location with a `MN` movement pattern to explore during learning, `FS NG` Panther has fixed start location but random exploration during learning, while `RS NG` Panther has random start locations and random exploration during learning. We make two main inferences from the experiment results:

1. *Exploration with self-play improves Panther scores against weaker or equal Pelican.* When Panther plays against a weaker Pelican than it trained against, or, trains using self-play while the Pelican it plays against also uses self-play during training, exploring the environment improves Panther scores. This can be attributed to the relative ineptness of the Pelican's weaker learned strategy that leaves holes for the Panther to escape if it explores the environment more effectively. For example, in the green-highlighted cells in Table 5, Panther wins nearly 100% when it is faced against a weaker Pelican that did not use learning. Similarly, for the blue-highlighted cells, Panther's wins improve to nearly 50% or more to overcome the game asymmetry while playing against an equal Pelican due to more exploration coupled with self-play.

2. *Exploration by Panther is not effective against a stronger Pelican.* When Panther plays against a stronger Pelican than it trained against (non-highlighted cells) without using self-play it performs poorly with around 10% wins. This is because, due to the asymmetric nature of the game, no matter how much exploration Panther does it has 'no way out' against the stronger Pelican. Finally, the orange-highlighted cells show that Panther with self-play performs poorly against a weaker Pelican. A possible explanation for this is that exploration along with self-play in these cases likely results in Panther overfitting its strategy to Pelican during training, making it less adaptable to different Pelican sonobuoy placement strategies during evaluation.

Clearly, an astute Pelican would train to be equal or stronger than the Panther, and, leave Panther at a disadvantage due to the asymmetry in the game. To alleviate this, we next evaluate different exploration and strategy diversity techniques for the Panther.

6.3 Evaluation of Different Panther Environment Exploration Strategies

For our next set of experiments, we evaluated the effect of improved Panther exploration of the environment (game board) on its learned evasion strategy while using three different initial movement patterns, Move-North, Diamond and Zig-Zag, described in Section 4.2. Recall that Panther's initial movement pattern enables it to explore the environment during its initial training phases and better exploration enables Panther to infer possible locations of Pelican's sonobuoys and evade them to improve its performance in the game. In each of the experiments, Pelican agents used are `Pel-RL-5`, that is, they use RL to train against Panther

Table 5: *Performance of different Panther models (rows) in terms of no. of wins out of* 100 *games/average rewards against different Pelican models (columns).*

	Pel-NO-3			Pel-RL-3			Pel-NO-5			Pel-RL-5		
Start location (*SS*) :	FS	FS	RS	FS	FS	RS	FS	FS	RS	FS	FS	RS
Move method (*MM*) :	MN	NG	NG	MN	NG	NG	MN	NG	NG	MN	NG	NG
Pan-RL-*SS*-*MM*-RL3	98	99	94	17	8	14	0	0	7	22	9	7
Pan-SP-*SS*-*MM*-RL3	26	17	17	18	45	64	1	3	0	52	41	67
Pan-RL-*SS*-*MM*-RL5	48	42	92	16	11	26	83	81	95	34	18	19
Pan-SP-*SS*-*MM*-RL5	97	4	33	13	48	58	39	0	0	7	47	47

Panther Fixed Start Loc. Panther Random Start Loc.

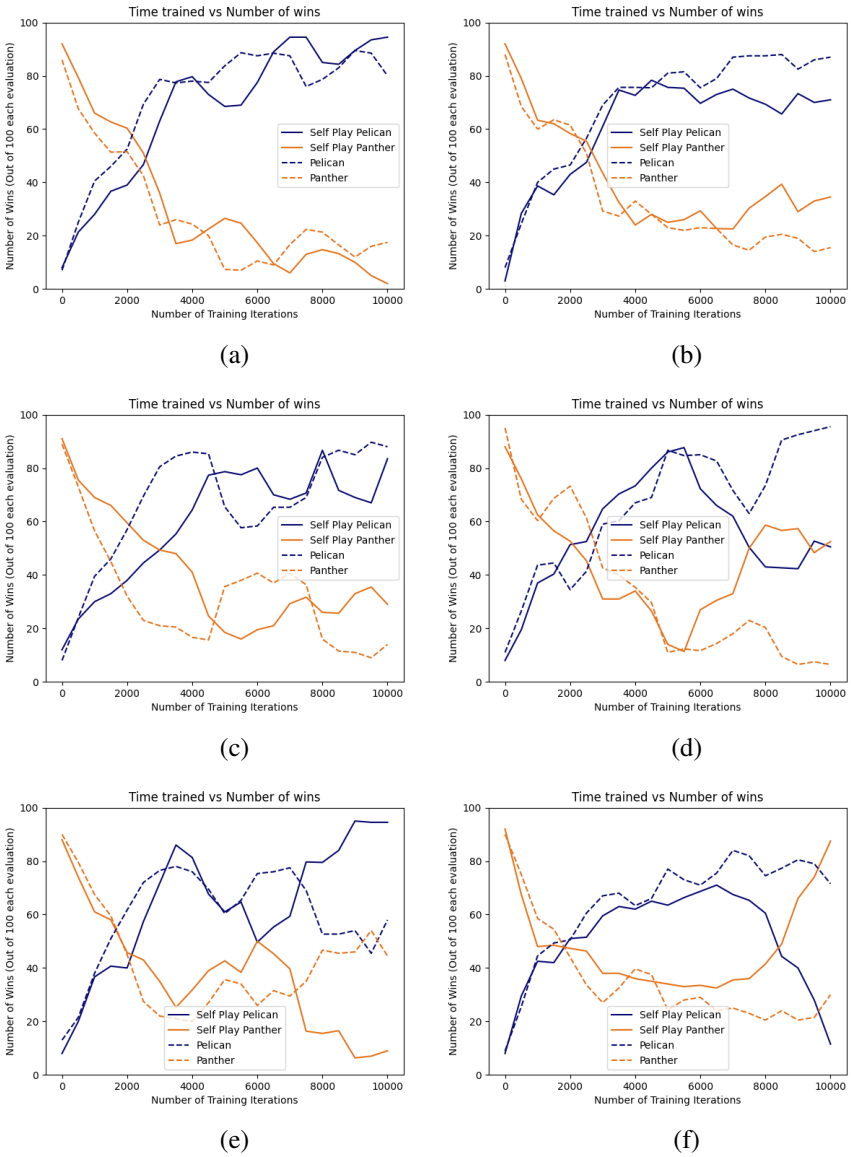


Figure 5: *Average wins and losses out of* 100 *games for Pelican and Panther versus no. of training iterations used to train the agents, with different Panther initial movement patterns, (a)-(b) Move North, (c)-(d) Diamond with radius*=2, 4, 6 *(e)-(f) Zig-Zag, with step-size*=2, 4, 6.

and can deploy up to 5 sonobuoys. Panther agents are varied over Pan–*AA*–*SS*–*MM*–RL5, where *AA*={RL, SP}, *SS*={FS, RS}, and *MM*={MN, MNR, DI,DIR, ZZ, ZZR}, giving 24 possible Pelican-Panther matchups. Other parameters used for the experiments are $RL_{algo} = \{A2C\}$, $t_{train} = 10^4$, $n_{bat} = 4$, $n_{test} = 100$ games. Pelican and Panther wins (out of 100 games) are recorded after every 1000 training iterations. Figure 5 shows the results of our experiments with dashed (solid) lines showing training without (with) self-play. Our results show that, as expected, the naive initial movement strategy Move-North (MN) is the least effective for Panther as it results in fewer than 20 wins with fixed start locations, as shown in Figure 5(a). When Panther starts at random locations using MNR pattern, in Figure 5(b), it makes the learning task harder for Pelican, and Panther's wins improve to about 20 without self-play and 40 with self-play. Diamond (DI and DIR) and Zig-zag (ZZ and ZZR) are more effective strategies than MN due to their improved exploration of the environment and they give notably higher wins than MN or MNR, as shown in Figures 5(c)-(f). Also, Panther's number of wins increases with fewer training iterations, indicating that the training takes less time to converge. We observe an interesting game dynamics in Figures 5(d)-(f): Panther and Pelican wins start to follow a cyclic pattern - as soon as each player learns a strategy to defeat its opponent decisively, the opponent too starts to learn a counter-strategy and soon 'turns the tables'. The cycle pattern in the number of wins continue until both players converge to the 50% win mark, which corresponds to the equilibrium outcome in the game.

6.4 Evaluation of Panther Strategy Diversity

For our final set of experiments, we investigated if strategy diversity via switching between different strategies could be used as a means to break the cycle pattern in number of wins of each player around the equilibrium. To clearly understand the effects of strategy diversity, we only allowed Panther agents to switch between different strategies while the Pelican agent used the same strategy, without switching, in all the experiments. Three Panther agents were first trained to play against a Pel–SP–5 Pelican yielding three Panther agents: Pan–SP–FS–MN–RL5, Pan–SP–FS–ZZ–RL5 and Pan–SP–FS–DI–RL5. A Pelican agent was then trained against the Panther using DI strategy, that is, Pan–SP–FS–DI–RL5. Note that because Pelican had specifically learned to play and defeat DI Panther during training, DI Panther got the least number of wins during testing. Each experiment was run for 100 games with the trained Pelican agent versus Panther switching between two or more of the trained strategies at intervals of 10, 20 and 30 games. Table 6 shows the results of the experiments. As illustrated by the results, Panther can improve its wins from a low-performing movement strategy, like DI, if it switches between the low-performing and another well-performing movement strategy, like MN. The switching interval was not found to have a significant effect on the number of wins of Panther, as shown by the low standard deviation values in the last column of Table 6. Overall, these results are useful because Panther is not aware which Pelican it is facing post-training, and it if maintains a collection of different strategies in its arsenal, it could switch between them and improve its wins if faced against a superior Pelican.

7 Conclusions and Future Work

Deep RL algorithms have been recently investigated as a suitable means for playing different RTS games and our experiences with evaluating and extending different deep RL algorithms for the HOTP game confirm this direction. Our main findings show that in a competitive setting like HOTP, any one player or team cannot learn a perennially winning strategy owing to the repeated, turn-taking nature of the game and players will repeatedly learn to

Table 6: *Number of Panther wins out of* 100 *games, averages and stdev for different Panther movement patterns without (top three rows) and with (bottom four rows) switching, with different Panther strategy switching intervals.*

Movement Pattern	Switching Interval			Ave.	Stdev
	10	20	30		
DI	16.8	16.8	16.8	16.8	0
ZZ	40.9	40.9	40.9	40.9	0
MN	57.7	57.7	57.7	57.7	0
ZZ ↔ DI	29.1	32.3	33.4	31.6	2.23
MN ↔ DI	38.3	41.5	44.1	41.3	2.91
MN ↔ ZZ	48.3	51.2	48.2	49.23	1.7
MN ↔ ZZ ↔ DI	40.1	38.8	39.4	39.43	0.65

defeat the opponent's strategy resulting in cycles around an equilibrium value of around 50% wins for each player. Using more diversity in movement strategies and switching between multiple learned strategies while playing the game also appears to be a suitable means to overcome the asymmetry of the game for the disadvantaged player. These results advance the state-of-the-art techniques in AI for playing RTS games providing approaches that could enable players to avoid repeated stalemate outcomes due to playing equilibrium strategies and improve their performance in the game. Our results pave the way for investigating many open problems and challenges in solving asymmetric RTS games like HOTP. RTS games have very large decision spaces and players also have partial or no information about opponent moves, and consequently, about the current state of the game before making a move. These challenges make it impractical to use state-of-the-art game-tree based search algorithms or vanilla deep RL algorithms as the only means to solve these games. It makes sense to develop theories and frameworks to investigate novel approaches that use clever tactics like focusing on only 'useful' parts of the game instead of the whole game board and adapting RL algorithms to learn quickly in large decision spaces. Recently proposed approaches like rainbow algorithms [31] that separately consider individual aspects from deep RL algorithms like deep Q networks to reduce the computation complexity, and Munchausen RL [20] that makes adjustment to the reward function to enable the RL agent to learn to solve a task rapidly and with fewer exploration indicate towards promising solutions to solve these issues. Applying these concepts in the context of RTS games is one of the future directions we are investigating. Another important direction in AI-based game playing techniques is to ensure that the decision made by players can be understood and analyzed by humans. We are exploring recently proposed techniques like saliency maps [22], history aware training [32] and graph attention-based techniques [33] in the context of the HOTP game to address this direction. Finally, hierarchical RL techniques [34] are a promising direction to investigate for Panther and Pelican to learn meta-strategies that could automatically determine when and which strategy each player could switch to, to improve its performance. Overall, the research in this paper provides a better understanding of some of the main challenges in asymmetric RTS games and we envisage that this direction will lead to more efficient techniques for solving RTS games and, more generally, for determining suitable strategies in competitive interactions between humans and human-machine teams.

Financial Support and Authorship Statement

This research was done as part of the Game Theoretic Machine Learning for Defense Applications project that was supported by a grant from Naval Research Laboratory 6.1 Base Funding Program. Dasgupta was responsible for conceptualizing and directing the research, and, for writing the paper. Kliem was responsible for implementing the code, running experiments and recording experimental results.

References

[1] M. S. Atkin, D. L. Westbrook, and P. R. Cohen, "Capture the flag: Military simulation meets computer games," in *Proceedings of AAAI Spring Symposium Series on AI and Computer Games*, 1999, pp. 1–5. [Online]. Available: https://www.aaai.org/Library/Symposia/Spring/1999/ss99-02-001.php

[2] K. Leune and S. J. Petrilli Jr, "Using capture-the-flag to enhance the effectiveness of cybersecurity education," in *Proceedings of the 18th Annual Conference on Information Technology Education*, 2017, pp. 47–52. [Online]. Available: https://doi.org/10.1145/3125659.3125686

[3] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman *et al.*, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019. [Online]. Available: https://doi.org/10.1126/science.aau6249

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1724-z

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nat.*, vol. 550, no. 7676, pp. 354–359, 2017. [Online]. Available: https://doi.org/10.1038/nature24270

[6] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "A survey on adversarial attacks and defences," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, pp. 25–45, 2021. [Online]. Available: https://doi.org/10.1049/cit2.12028

[7] E. Löffler, B. Schneider, T. Zanwar, and P. M. Asprion, "Cysecescape 2.0 - A virtual escape room to raise cybersecurity awareness," *Int. J. Serious Games*, vol. 8, no. 1, pp. 59–70, 2021. [Online]. Available: https://doi.org/10.17083/ijsg.v8i1.413

[8] M. Salovaara-Hiltunen, K. Heikkinen, and J. Koivisto, "User experience and learning experience in a 4d virtual reality simulation game," *Int. J. Serious Games*, vol. 6, no. 4, pp. 49–66, 2019. [Online]. Available: https://doi.org/10.17083/ijsg.v6i4.305

[9] L. Hanes and R. Stone, "Applying constrained virtual environments to serious games for heritage," *Int. J. Serious Games*, vol. 6, no. 1, pp. 93–116, 2019. [Online]. Available: https://doi.org/10.17083/ijsg.v6i1.294

[10] V. G. Goecks, N. Waytowich, D. E. Asher, S. J. Park, M. Mittrick, J. Richardson, M. Vindiola, A. Logie, M. Dennison, T. Trout, P. Narayanan, and A. Kott, "On games and simulators as a platform for development of artificial intelligence for command and control," *The Journal of Defense Modeling and Simulation*, p. 15485129221083278, 2022. [Online]. Available: https://doi.org/10.1177/15485129221083278

[11] W. Ruan, H. Duan, and Y. Deng, "Autonomous maneuver decisions via transfer learning pigeon-inspired optimization for ucavs in dogfight engagements," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 9, pp. 1639–1657, 2022. [Online]. Available: https://doi.org/10.1109/JAS.2022.105803

[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012. [Online]. Available: 10.1109/TCIAIG.2012.2186810

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[14] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: 10.1126/science.aar6404

[15] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012. [Online]. Available: 10.1109/TCIAIG.2012.2200894

[16] J. Wang, T. Zhu, H. Li, C.-H. Hsueh, and I.-C. Wu, "Belief-state monte-carlo tree search for phantom games," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 267–274. [Online]. Available: 10.1109/CIG.2015.7317917

[17] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018. [Online]. Available: https://doi.org/10.1126/science.aao1733

[18] B. Cui, H. Hu, L. Pineda, and J. Foerster, "K-level reasoning for zero-shot coordination in hanabi," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8215–8228, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2207.07166

[19] S. Huang, S. Ontañón, C. Bamford, and L. Grela, "Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning," in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–8. [Online]. Available: 10.1109/CoG52621.2021.9619076

[20] N. Vieillard, O. Pietquin, and M. Geist, "Munchausen reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4235–4246, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2007.14430

[21] J. Robertson, A. V. Kokkinakis, J. Hook, B. Kirman, F. Block, M. F. Ursu, S. Patra, S. Demediuk, A. Drachen, and O. Olarewaju, "Wait, but why?: assessing behavior explanation strategies for real-time strategy games," in *26th International Conference on Intelligent User Interfaces*, 2021, pp. 32–42. [Online]. Available: https://doi.org/10.1145/3397481.3450699

[22] A. Anderson, J. Dodge, A. Sadarangani, Z. Juozapaitis, E. Newman, J. Irvine, S. Chattopadhyay, M. Olson, A. Fern, and M. Burnett, "Mental models of mere mortals with explanations of reinforcement learning," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 10, no. 2, pp. 1–37, 2020. [Online]. Available: https://doi.org/10.1145/3366485

[23] J. D. Salt, "Hunting of the plark: A game of anti-submarine warfare," Cranfield University, Tech. Rep., 2015.

[24] Montvieux, "The hunting of the plark: Code repository," https://github.com/montvieux/plark-ai-public/, 2020, accessed: 2021-08-12.

[25] I. Kargar, *Visual Explanation of the A2C reinforcement learning algirithm*. Medium.com, Nov 2019. [Online]. Available: https://kargarisaac.medium.com/rl-series-a2c-and-a3c-in-pytorch-6e9edf5c8788

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018. [Online]. Available: https://dl.acm.org/doi/10.5555/3312046

[27] L. Graesser and W. L. Keng, *Foundations of deep reinforcement learning: theory and practice in Python*. Addison-Wesley Professional, 2019.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1707.06347

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1602.01783

[31] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018. [Online]. Available: https://doi.org/10.1609/aaai.v32i1.11796

[32] M. Wright, Y. Wang, and M. P. Wellman, "Iterated deep reinforcement learning in games: History-aware training for improved stability." in *EC*, 2019, pp. 617–636. [Online]. Available: https://doi.org/10.1145/3328526.3329634

[33] W. J. Yun, S. Yi, and J. Kim, "Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 2967–2972. [Online]. Available: 10.1109/SMC52423.2021.9658625

[34] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019. [Online]. Available: https://doi.org/10.1007/s10458-019-09421-1