Navigation in Adversarial Environments Guided by PRA* and a Local RL Planner

by

Debraj Ray

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

© Debraj Ray, 2023

Abstract

Real-time strategy games require players to respond to short-term challenges (micromanagement) and long-term objectives (macromanagement) simultaneously to win. However, many players excel at one of these skills but not both. This research studies whether the burden of micromanagement can be reduced on human players through delegation of responsibility to autonomous agents. In particular, this research proposes an adversarial navigation architecture that enables units to autonomously navigate through places densely populated with enemies by learning to micromanage itself. Our approach models the adversarial pathfinding problem as a Markov Decision Process (MDP) and trains an agent with reinforcement learning on this MDP. We observed that our approach resulted in the agent taking less damage from adversaries while travelling shorter paths, compared to previous approaches on adversarial navigation. Our approach is also efficient in memory use and computation time. Interestingly, the agent using the proposed approach outperformed baseline approaches while navigating through environments that are significantly different from the training environments. Furthermore, when the game design is modified, the agent discovers effective alternate strategies considering the updated design without any changes in the learning framework. This property is particularly useful because in game development the design of a game is often updated iteratively.

Acknowledgements

I am grateful to Professor Nathan Sturtevant for supervising the work, offering invaluable advices in all the stages of the research, and being patient with me. I am also thankful to him for sharing a lot of resources online, such as the MovingAI.com website, the HOG2 repository, and lecture recordings on YouTube, all of which I have referred to frequently during my research.

I am also thankful to my defense committee for their expert insights and comments on my work. Furthermore, I acknowledge the wonderful pedagogy and platform the University of Alberta has provided me which made this research possible.

I thank Antonie Bodley for her careful evaluation of my writing and feedback on grammar and readability.

Last but not least, I thank my wife for always keeping me motivated.

Table of Contents

1 Motivation **1**

1.0.1 Moving Units through Hostile Areas in Video Games 1

1.1 Adversarial Navigation in the Real World, Autonomous Package Delivery 4

2 Introduction **5**

2.1 Thesis Objectives 7

3 Related Work **10**

3.1 Machine Learning Approaches to Pathfinding 12

4 Background **14**

4.1 Reinforcement Learning 14

4.1.1 Deep Q-Network 16

4.2 Partial Refinement with Abstraction and A* (PRA*) 17

4.3 Potential Fields 18

5 A Novel Approach to Optimizing Adversarial Navigation with RL **21**

5.1 Defining Adversarial Navigation 21

5.2 Our MDP For Adversarial Navigation 22

5.3 Encoding Objectives in the Rewards Function 23

5.4 Q-Network Architecture 25

6 Environmental Representations **29**

6.1 Risk Representation 30

6.2 Rotation Abstraction 31

6.3 Simple Grid-based Raycast Distances 34

6.4 Path Representation 34

7 Experimental Results **37**

7.1 Experiment 1 : Adversarial Navigation on the Warcraft III Maps . . . 40

7.2 Experiment 2 : Adversarial Navigation on the Warcraft III Maps After a Game Design Update. 46
7.3 Experiment 3 : Adversarial Navigation on the Training Maps 50
7.4 Limitations of the Proposed Approach 53

8 Conclusion **55**

Bibliography **57**

List of Tables

6.1 Function to normalize angular distance of enemies 30

7.1 Rewards used to train the agent 39
7.2 Number of enemies in Warcraft III maps 41
7.3 Mean and 95% confidence intervals of damage and path ratio in experiment 1. 45
7.4 Mean and 95% confidence intervals of damage and path ratio in experiment 2 having updated game design 50

List of Figures

1.1 A snapshot from Age of Empires 1. War elephants under attack from enemy towers. 2
1.2 A snapshot from StarCraft 2. Allied units under attack from the enemies en-route. 3
1.3 A screenshot from Dota 2. Units encountered adversaries on the way. 3

4.1 The agent environment interaction in reinforcement learning as a Markov Decision Process (Sutton and Barto, 2018). 14
4.2 Abstracting a general path (Sturtevant and Buro, 2005). 17
4.3 Composite potential field of target and obstacles.(Jiang, Cai, and Xu, 2023). 19

5.1 A model of the neural network for learning navigation in adversarial environments. 26

6.1 Plot of the normalizing function $f(\theta)$ 30
6.2 Change of coordinate system for rotation abstraction. 31
6.3 Possible positions of the agent along the path is shown. Path directions are marked with red arrows. Agent's direction is inferred and shown with black arrows. 35

7.1 A generated map used to train the agent. 37
7.2 Warcraft 3 maps used to test the agent. From left to right: blastlands, divfideandconquer, duskwood, gardenofwar, and thecrucible. 40
7.3 Amount of damage the agent takes, plotted against optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors 42
7.4 Path length suboptimality for different types of agents. 43

7.5 The maximum memory used for path finding measured by the size of the open list, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors 43
7.6 The amount of time in milliseconds that the agent took to reach the destination, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors 45
7.7 Amount of damage the agent takes in experiment 2, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors 47
7.8 Path length suboptimality for different types of agents in experiment 2. 48
7.9 The maximum memory used for path finding measured by the size of the open list in experiment 2, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors 48
7.10 The amount of time in milliseconds that the agent took to reach the destination in experiment 2, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors 49
7.11 Amount of damage the agents take in experiment 3. 51
7.12 Path length sub-optimality for different types of agents in experiment 3. 52
7.13 The maximum memory used by the agents for navigation in experiment 3. 52
7.14 The amount of time in milliseconds that the agents took to reach the destination in experiment 3. 53

Chapter 1

Motivation

Real-time strategy (RTS) games are a genre of strategy games requiring players to make decisions and moves simultaneously in real time. RTS games differ, for example, from combinatorial games such as Chess or Checkers which are sequential in nature, i.e. players take turns to make moves. Players of RTS games are typically challenged with several objectives, such as gathering resources, engaging in battles, territorial expansion, economic and technological advancement, etc. In order to secure victory, a player needs to outmaneuver the opponents and accomplish certain overarching goal like destroying opponent's key infrastructure, or defending a base for certain duration.

This section lists a few instances from RTS games that demonstrate motivation for this research.

1.0.1 Moving Units through Hostile Areas in Video Games

Imagine four war elephants are navigating to a battle site (Figure 1.1). On the way, they come under attack from two enemy watch towers. The war elephants have the option to counter attack the towers. Or, they may ignore the towers and continue. The human player may, in this situation, instruct one war elephant to engage the enemies and the other three war elephants to move towards the battle site. Sacrificing a specific unit to protect other important units (such as heroes) is sometimes an effective strategy in adversarial navigation. While micromanaging navigation is essential, it requires significant time and effort from the human player.

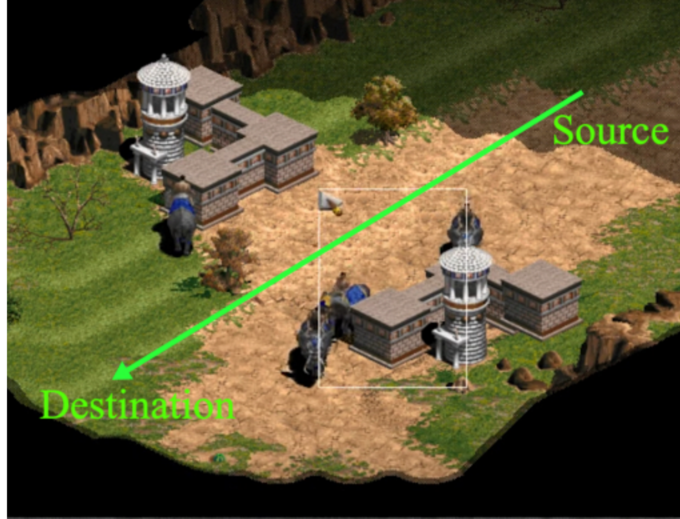

Figure 1.1: A snapshot from Age of Empires 1. War elephants under attack from enemy towers.

We study the question whether we can make the units intelligent by default so that the human players have more time and attention to develop higher-level complex strategies. This can potentially evolve RTS game dynamics with greater focus on macromanagement. This research is a step towards realizing this goal.

Figure 1.2 is a similar situation where allied troops are intercepted by enemy units. In this specific situation, the enemy units are stronger than the allies and therefore attacking the enemies directly is dangerous. Alternately, the allies can trick the enemies by moving close to the enemies and then quickly retreating. As a result, some of the enemies would leave the group to chase the allies. Once the chasing-enemies are at a safe distance from the larger group, the allies can quickly attack and destroy them. This will weaken the remaining enemies of the group. The allies can then launch a successful attack. This situation is another example when micromanagement of navigation is crucial for the safety of the units. Through this research, we propose to make the units intelligent and reduce the burden of micromanagement on the human players.

In a gameplay instance from Dota 2 (Figure 1.3), units proceeding toward a destination have inadvertently come under attack from enemies. An enemy unit is hurling

Figure 1.2: A snapshot from StarCraft 2. Allied units under attack from the enemies en-route.

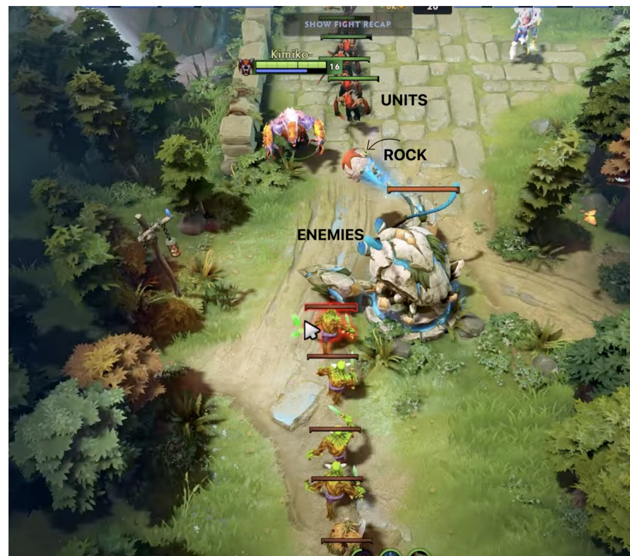

Figure 1.3: A screenshot from Dota 2. Units encountered adversaries on the way.

giant rocks. In this case, the enemy is immobile, and the speed of the attack is slow. Therefore, it would be prudent not to engage this enemy and instead move quickly out of the enemy's attack range.

Navigation through hostile environments should consider the influence of adversaries while simultaneously path planning for realistic behavior and unit longevity. Consider a situation where fishing boats come under attack from enemy ships. If the fishing boats have to wait for a human player to micromanage a rescue plan, important time could be lost, resulting in higher overall damage. Alternatively, the fishing boats could be intelligent by default and autonomously sail to a safe harbor where allied battleships can protect them.

This research explores the idea of autonomous navigation in adversarial environments as a way to lower the burden of micromanagement on human players and potentially evolve gameplay experience.

1.1 Adversarial Navigation in the Real World, Autonomous Package Delivery

While this research is useful in virtual settings, it also has implications for real-world applications. Delivery bots on the road from companies like Starship and Nuro are increasingly common. These robots are susceptible to adversarial attacks (such as theft or vandalism) from people and animals on the road, particularly in dangerous neighbourhoods. Ideas from this research can be used to enhance the robot's intelligence for quick and safe transit through these areas.

Chapter 2

Introduction

In RTS games, one common metric predictive of player performance is actions per minute (APM) (Avontuur, Spronck, and Van Zaanen, 2013). A high APM rate often means the player is micromanaging the behaviour of all the units under their control to get the best possible behavior. While expert players do not necessarily want games to reduce their burden of micromanagement since it gives them an advantage (Anhalt, 2011), one could imagine many scenarios where players could find increased enjoyment when their units have better default behaviors. In particular, players want units to trade off path-length optimality for other objectives. For example, offensive RTS units might collaborate to encircle the enemy, while defensive units may prefer safer escape routes (Botea et al., 2013). Other strategies for safe navigation include keeping distance from enemies and avoiding enemy vantage points and areas commonly patrolled by enemy scouts to avoid detection (Critch and Churchill, 2020). Makarov, Polyakov, and Karpichev (2018) observed that the construction of Voronoi-based navigation mesh in a first-person shooter game for tracking tactical properties like line of fire, kill / death statistics, and enemy visibility significantly improved win-rate. Tremblay, Torres, and Verbrugge (2014) showed the benefit of strategic navigation for stealth operations in games. Ninomiya et al. (2015) described a constraint-aware path-planning system that ensures minimum distance traveled from source to destination while satisfying certain constraints. Constraints can be location specific, such

as the unit needs to check behind buildings. Or it can be about enemy evasion, such as requiring the unit to avoid being seen. Constraints can also relate to unit organization, for example, the unit should stay in between two other units. A dynamic path planner initially computes the constraint-aware path and later repairs it if constraints are updated due to changing environmental dynamics.

In previous research, Levy et al., 2020 used deep reinforcement learning to generate safe routes in urban environments while optimizing path lengths. However, this work learns features from the entire map, which does not generalize well to unseen maps or scale to maps of arbitrary sizes. Other works on adversarial navigation use influence maps (Critch and Churchill, 2020) and potential fields (Hagelbäck, 2016). PRA* (Sturtevant and Buro, 2005; Sturtevant et al., 2019) is an efficient pathfinding and navigation technique well-suited for dynamic environments, including hostile ones. The is because in a dynamic environment computed paths get outdated quickly. Since PRA* works by cheaply computing abstract paths with only partial refinement of the real path, it can afford to replan repeatedly to avoid local enemies. The previous approaches, however, require manual tuning of parameters that determine the magnitude of the agent's attraction to and repulsion from objects in the environment.

With the broader goal of better micromanagement in mind, this thesis studies the specific problem of adversarial pathfinding, proposing a reinforcement-learning (RL) based approach combined with the PRA* for hierarchical pathfinding. Reinforcement learning (RL) is used for learning a policy to counter adversaries during local movement while also optimizing path suboptimality. The agent then uses PRA* for long-distance pathfinding, considering only static obstacles, using local strategies learned through RL to reach nearby goals. While PRA* is a proven better alternative to A* for pathfinding in dynamic environments, RL reduces the burden of refinement on PRA* at the lowest levels of abstraction (that considers dynamic obstacles and adversaries), thus resulting in a mutually beneficial relationship. Furthermore, our use of environmental abstractions enables the agent to effectively execute adversarial

navigation on maps that are significantly different from the maps used to train the agent. It is also observed that the proposed approach is generalizable in the sense that the agent can learn different strategies when the game design is updated without requiring any code changes. This is a desirable property because game design is usually an iterative process (Macklin and Sharp, 2016).

Experimental results show that agents using the proposed approach take about four times less damage than baseline approaches based on either potential fields or PRA* alone while not requiring significantly more computation resources (memory and CPU time). Furthermore, the proposed approach is shown to scale to different kinds of environments and those of varying sizes.

2.1 Thesis Objectives

We propose an RL based framework for autonomous navigation in adversarial environments that minimize path suboptimality and damage from adversaries in the environment. We compare our approach with other baseline approaches on these metrics: path length from source to destination, the amount of damage the agent takes, amount of memory used for pathfinding, and the time to reach the destination. Experimental results show that our approach outperforms the baseline approaches in path length optimality and unit safety while not compromising on system resources - memory and CPU time.

We are also interested in the generalizability of our approach. At any instant, the agent has visibility only of the portion of the map within its field of view (FOV). This makes the approach scalable because the original map can be of any size, and yet the agent can perform local adversarial navigation. Our learning framework uses a single convolution layer to extract features from the agent's FOV coupled with a set of computed abstract features. The simple structure of the neural network powered by domain knowledge (used to define the abstract features) makes the generated model highly explainable. In turn, this improves the model's generalizability because we have

good control over what is being learnt, allowing us to curate features that generalize across environments. Ba and Caruana (2014) have shown that shallow networks are not just faster to train than deep networks but also learn equally good representations. As a result, the agent trained on small custom maps (size 27×27) could successfully navigate large maps (size 512×512) of the popular RTS game Warcraft III even when the test maps are significantly different from the training maps.

The proposed approach can uncover alternate strategies autonomously when the game design is updated without changes to the learning framework. This is essential because game designing is normally an iterative process. However, there are limitations to this generalization. For example, if a major design update occurs, the agent's state representation could become insufficient. Then an update to the learning framework may be needed.

Our contributions are as follows:

- We introduced a novel approach for adversarial navigation that combines PRA* and RL.
- We show that the approach can generalize to different game environments, especially standard game maps that have not been used to train the agent.
- We show that the approach can scale to maps of different sizes.
- We show that the approach is generalizable across game designs. That is, if the game design is updated, the agent can uncover new strategies for adversarial navigation following the proposed approach, without requiring update to the agent's learning framework.
- We provide a comparison to current state-of-the-art approaches for adversarial navigation in terms of path length suboptimality, damage to the agent during navigation, time to reach the destination, and the amount of extra space used.

This work has been accepted to The 19th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-2023).

Chapter 3

Related Work

This chapter discusses in detail previous work on tactical navigation, methods that were used to overcome adversaries during navigation, and limitations of those approaches. It further discusses Machine Learning based approaches for mutiobjective path finding and navigation in video games and mobile robots.

Real-time strategy games usually take place in dynamic and hostile environments where players have to make high-level and low-level decisions under extreme time constraints (Lara-Cabrera, Cotta, and Fernández-Leiva, 2013; Sailer, Buro, and Lanctot, 2007). Hagelbäck (2016) proposed a hybrid navigation system for the RTS game StarCraft combining potential fields with A*. Their paper states that A* alone is not well-suited for dynamic worlds because the environment can change as the unit navigates, and the path computed by A* could quickly become obsolete. An alternate approach called PRA* (Sturtevant and Buro, 2005) involves constructing hierarchical abstractions on top of a given map and then partial pathfinding through refinement. Unlike A*, PRA* is suited for dynamic environments because it interleaves partial path planning and path execution, which helps solve the problem of long paths becoming obsolete over time. This is because the last stage of partial pathfinding does not need to happen until an agent is nearby, meaning that the local environment is known with higher certainty.

In the field of adversarial pathfinding, stealth games like Tom Clancy's Splinter

Cell (Ubisoft, 2002), Mark of the Ninja (Microsoft Studios, 2012), and Metal Gear Solid V (Konami, 2015) encourage players to follow covert paths and utilize light and sound to avoid enemy detection during navigation. Mendonça, Bernardino, and Neto (2015) proposed a method to construct such covert paths in real-time using navigation meshes. Other researchers have attempted to strike a balance between path suboptimality and path safety in adversarial navigation. Jong et al. (2015) used influence maps (IMs) to compute a safety score for each grid cell of the map and then used it in the heuristic function of A* to find safe paths. The paper observed that the search space of A* increases significantly due to this additional dimension. The paper then used PRA* to improve search performance. A similar idea based on influence maps and flocking was implemented by Danielsiek et al. (2008) for group navigation of units in the RTS game Glest. Avery, Louis, and Avery (2009) used a set of influence maps to execute complex tactics in a naval RTS game having an adversarial environment. Influence maps require storing of influence values for every entity of the world and then use a pathfinding algorithm such as Dijkstra or A* to find an optimal path by incorporating the influence values in the cost function (Adaixo, 2014). Additionally, influence maps are computationally expensive without parallelization since they must be updated frequently when the position of units in the world changes (Mark, 2019). With parallelization, the implementation complexity of the algorithm increases considerably.

Stanescu, Barriga, and Buro (2014) discussed a method to decompose strategies in RTS games into decoupled hierarchical abstractions. The first layer deals with the highest level strategies to win the game, the second layer encapsulates action plans to execute the higher level strategies, and the third layer's task is to generate the actual execution plan in terms of real-world moves. The authors showed this approach outperformed other state-of-the-art algorithms like UCT and Portfolio Search in large combat experiments using StarCraft.

The game of capture-the-flag (CTF) is interesting in the context of adversarial

pathfinding as it requires complex strategic navigation. Players need to defend their own flag while trying to steal their opponent's flag. Huang et al. (2011) calculated the winning regions of both the attacker and the defender using the Hamilton-Jacobi reachability. Like CTF, the game of cops and robbers requires adversarial navigation. Moldenhauer and Sturtevant (2009) explored different algorithms for the robber to evade capture by the cop for as long as possible. Tastan, Chang, and Sukthankar (2012) analyzed the problem of intelligent interception of adversaries in first person shooter games. The authors modelled the adversary's motion using inverse reinforcement learning and used that knowledge to track and ambush the adversaries at specific target points. Besides video games, tactical adversarial navigation has applications in the real world too. For example, a wildlife security group (PAWS) optimized its use of human resources for patrolling large conservation areas while effectively combating poaching (Fang et al., 2016).

Wang et al. (2020) proposed a method to navigate multiple mobile robots in a dynamic environment. The method uses RL to avoid collisions with static and dynamic obstacles while following a global guidance path computed using the A* algorithm. This work, however, has not addressed the problem of adversarial navigation and did not demonstrate whether their approach can generalize to different types of maps with different environment dynamics. Our work extends the idea to adversarial environments, explores PRA* as an alternate global guidance, and is generalizable.

3.1 Machine Learning Approaches to Pathfinding

Panov, Yakovlev, and Suvorov (2018) explored a RL based pathfinding technique called Neural-Q learning. It uses a reward structure that benefits the agent if its action moves it closer to the destination. Moreover, the authors used a regularization technique that penalizes the agent for straying off the path. The authors reported that, after training, the agent could successfully navigate both seen and unseen paths. However, there were instances when the agent failed to find any path (even though

one exists) or the path that it found was longer than the one found with A*.

Navigation meshes (NavMeshes) are the industry standard for NPC navigation in video games. The resultant graph is a compact representation of the 3D environment as complex polygons. These graphs, however, do not incorporate the NPC's unique abilities. Traditional solutions do not make full use of the NPC's abilities and often look unrealistic. Alonso et al., 2020 demonstrated a solution with deep reinforcement learning that worked well and eliminated the need for manually modifying the NavMesh, thus achieving major cost saving procedurally generated for large maps.

Neural A* is a differentiable formulation of the A* search algorithm coupled with a convolutional encoder (Yonetani et al., 2021). It benefits in path length optimality by encoding visual information as guidance maps which then guide the A* search. The module performs A* search in the forward pass and backpropagates losses. The loss is computed by comparing against ground-truth optimal paths provided either by human annotators or the neural planner. The objective of loss minimization is to reduce the inconsistency between resulting search histories and ground truth paths. The authors showed that Neural A* could predict realistic human trajectories on natural images provided as input.

Multi agent path finding (MAPF) is a well-known planning problem that requires multiple agents navigating in a common environment to reach their respective destinations without collision (Stern et al., 2019). A solution to MAPF was explored with reinforcement and imitation learning. The technique named PRIMAL taught the agents fully decentralized policies such that the agents can implicitly coordinate to avoid collisions while planning paths in a partially observable environment (Sartoretti et al., 2019). However, MAPF usually involves cooperative agents but this work focusses on adversarial agents.

Chapter 4

Background

4.1 Reinforcement Learning

RL entails an agent observing a state of the environment S_t at timestep t and responding with an action A_t (Figure 4.1). Consequently, the environment provides the agent a scalar reward R_{t+1} and transitions the agent to the next state S_{t+1} at time $t + 1$. The agent's objective is to learn an optimal policy π^* that maximizes the expected discounted future rewards (Arulkumaran et al., 2017). Low values of the discounting factor γ $[0, 1]$ gives more importance to immediate rewards. The policy π is a mapping from the states S to a probability distribution over actions, $\pi : S \rightarrow p(a = A|S)$. The optimal policy ($\pi_*$) is defined as:

$$\pi_* = \arg\max_{\pi}(\mathbb{E}[R|\pi]) \tag{4.1}$$

An RL problem can be formally described as a Markov Decision Process (MDP)

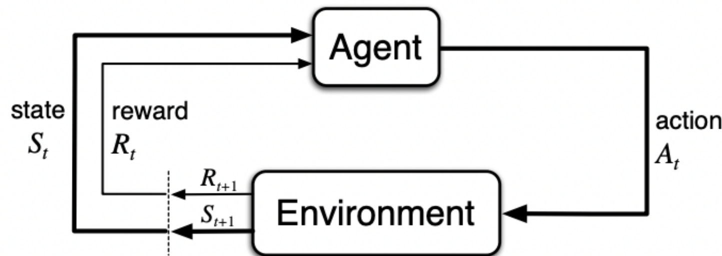

Figure 4.1: The agent environment interaction in reinforcement learning as a Markov Decision Process (Sutton and Barto, 2018).

represented as a tuple $< S, A, P, R >$ (Puterman, 2014). Here S represents a finite set of states and A represents a finite set of actions. P is a set of transition probabilities, where $P(s'|s, a)$ is the probability of transitioning from state s to s' after taking action a. R is a reward function, where $R(s, a)$ depends on the state and action. In RL, an important assumption is the Markov Property which states the future is conditionally independent of the past if the current state is known. In other words, the decision to take at state s_{t+1} can be made based solely on s_t. However, this assumption is often unrealistic because it requires the states to be fully observable.

A value function represents the state's expected cumulative discounted future rewards and is a measure of how good or bad each state is (Li, 2018). Under policy π, a state value is defined formally as $v_\pi(s) = \mathbb{E}_\pi[R_t|S_t = s]$. The Bellman equation for state value is (Sutton and Barto, 2018) :

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r \ + \ \gamma v_\pi(s')] \tag{4.2}$$

The optimal state-value is $v_*(s) = \max_\pi v_\pi(s)$. The Bellman equation for $v_*(s)$ is provided below (Sutton and Barto, 2018).

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r \ + \ \gamma v_*(s')] \tag{4.3}$$

The optimal action value $q_\pi(s, a) = \mathbb{E}[R_t|S_t = s, A_t = a]$ is the expected return when selecting action a in state s, then following policy π. The Bellman equation for action-value is (Sutton and Barto, 2018) :

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)[r \ + \ \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \tag{4.4}$$

The Bellman equation for the optimal action value is (Sutton and Barto, 2018) :

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r \ + \ \gamma \max_{a'} q_*(s',a')] \tag{4.5}$$

4.1.1 Deep Q-Network

In the DQN approach, a deep neural network (also called the Q-Network) is used to approximate the action-value function. A network with L (where $L > 0$) hidden layers, represented as $\{d_i\}_{i=0}^{L+1} \subseteq \mathbb{N}$ approximates a function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{L+1}}$ (Fan et al., 2020).

$$f(x) = W_{L+1}\ \sigma(W_L\ \sigma(W_{L-1}\ \sigma(\ldots(W_2\ \sigma(W_1 x + b_1) + b_2)\ldots) + b_{L-1}) + b_L) + b_{L+1} \tag{4.6}$$

In (4.6), $W_l \in \mathbb{R}^{d_l X d_{l-1}}$ is the weight matrix, and $b_l \in \mathbb{R}^{d_l}$ is the bias vector of the lth layer of the neural network. A non-linear transformation σ is applied to each layer. The network parameters are the weights and biases. They are learned in the process of training.

The Q-network is denoted as $Q(s, a; \theta)$, where θ represents the network parameters. It is trained by minimizing a sequence of loss functions $\delta_i(\theta_i)$ at every iteration i of the training (Mnih et al., 2013).

$$\delta_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}[(y_i - Q(s,a;\theta_i))^2]$$

where,

$$y_i = \mathbb{E}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}|s,a)]$$

The target for the next iteration is represented as y_i. $\rho(s, a)$ is the behaviour policy. In DQN, the target is a moving estimate of the actual Q value rather than being fixed in the beginning, as is usually done for supervised learning. This concept is known as bootstrapping. The loss function is optimized with stochastic gradient descent. The derivative of the loss function with respect to θ is given below.

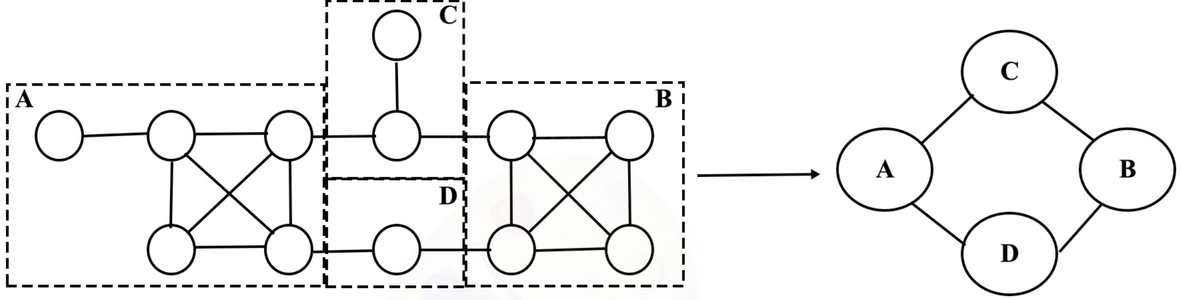

Figure 4.2: Abstracting a general path (Sturtevant and Buro, 2005).

$$\nabla_{\theta_i}\delta_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.);s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)]\nabla_{\theta_i}(Q(s, a, \theta_i)) \quad (4.7)$$

4.2 Partial Refinement with Abstraction and A* (PRA*)

A navigation task can be decomposed into two parts: path finding and path execution. The traditional approach first computes the entire path from source to destination and then requires the agent to follow or execute the path. Sturtevant and Buro, 2005 suggested an alternate idea called Partial Refinement A* (PRA*). Instead of computing the entire path from source to destination, a high level plan is initially created, and then the actual path is computed partially and progressively as the agent travels. The idea interleaves path computation with path execution which makes it robust to environment dynamics and precludes path re-computation.

The first step involves creating an abstraction of the world where a vertex in the abstract space represents a group of vertices in the real world. A "clique and orphan" abstraction pattern ensures that all real nodes of an abstract node are reachable from each other in one step. An orphan node in abstract space can be reached only from a single real node. In Figure 4.2, the nodes in the real world are abstracted into cliques marked A, B, C, D based on their connectivity to neighbouring nodes. The cost of constructing abstraction is $O(n)$, where n is the number of vertices of a graph.

PRA* performs an A* search in the abstract space using an abstract heuristic

function. The search cost in the real world is reduced by ensuring that A* search in the real world generates only those nodes whose parents are part of the abstract path. This constrained A* search is the primary reason for PRA*'s significant speedup over A* while also utilizing less memory than A*.

While the original PRA* paper used multiple abstraction levels, we have used a single abstraction level in this research.

4.3 Potential Fields

The idea of potential fields is about using attractive and repulsive forces to guide the agent's navigation. In RTS games, the attractive charge is placed at the goal (or position to be reached), and repulsive charges are on obstacles. The magnitude of these repelling charges is typically small and within a short radius of the obstacle, while the magnitude of the attractive force of the destination is large and must reach the entire virtual world. The potential field from adversaries can depend on many factors, such as if the agent can attack the opponent, the opponent can attack the agent, whether the agent or the opponent is on the offensive or defensive, and the range of the weapon of the agent and the opponent (Hagelbäck, 2012). A problem with potential field based navigation is that the agent might get stuck in local optima. This is a point of high potential surrounded by regions of low potential and is not the destination. A solution proposed by Hagelbäck and Johansson (2021) used a repulsive trail (also called potential trail) to push the agent forward and prevent it from coming back to its earlier locations. This approach was successful in preventing the agent from getting stuck in local optima when deployed in the open source RTS game, ORTS (Buro and Churchill, 2012).

We represent a 2D world $W \subseteq \mathbb{R}^2$. The current location of the agent in W is q. Then the potential field U(q) is calculated by superimposing the attraction and repulsion fields at q (Filimonov, Filimonov, and Barashkov, 2019) (Figure 4.3).

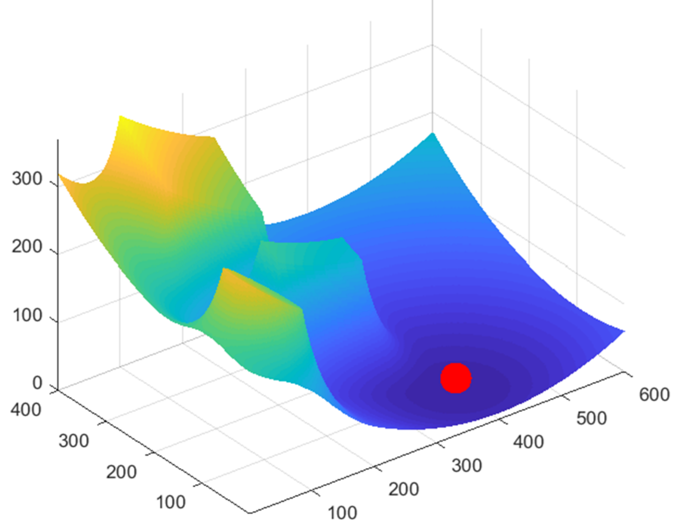

Figure 4.3: Composite potential field of target and obstacles.(Jiang, Cai, and Xu, 2023).

$$U(q) = U_{attractive}(q) + U_{repulsive}(q) \tag{4.8}$$

The potential force F(q) is calculated as below.

$$F(q) = -\nabla U(q) = -\begin{bmatrix} \frac{\partial U(q)}{\partial x} & \frac{\partial U(q)}{\partial y} \end{bmatrix} \tag{4.9}$$

The attractive potential is usually a parabolic function.

$$U_{attractive}(q) = \frac{1}{2} K_a . d(q)^2 \tag{4.10}$$

Here $d(q)$ is the Euclidean distance of the agent from the goal, $d(q) = ||q - q_{\text{goal}}||$. K_a is a constant, such that $K_a > 0$. The repulsive potential is calculated as shown below.

$$U_{repulsive}(q) = \begin{bmatrix} \frac{1}{2} K_r \cdot (\frac{1}{\rho(q)} - \frac{1}{\rho_0})^2, \ \rho(q) \leq \rho_0 \\ 0, \ \rho(q) > \rho_0 \end{bmatrix} \tag{4.11}$$

Here, $\rho(q)$ is the minimum distance of the agent to the obstacles. This formulation creates a repulsive field of radius ρ_0 around the obstacle. An important disadvantage

of this solution is that the attractive force decreases quickly and becomes nearly zero when the agent is close to the goal. To overcome this problem, the definition of attractive forces is modified slightly.

$$U_{attractive}(q) = \begin{bmatrix} \frac{1}{2}K_a \cdot d(q)^2, \; d(q) \leq d_0 \\ \frac{1}{2}K_a \cdot d(q), \; d(q) > d_0 \end{bmatrix} \tag{4.12}$$

Here, d_0 is assumed to be a short distance from the goal.

In this chapter we reviewed the literature on RL, focussing on the DQN algorithm. For our approach we chose DQN because of the algorithm's inherent simplicity and easy implementation. Next, we reviewed two methods: PRA* and Potential Fields, for executing adversarial navigation in dynamic environments. Since these are popular approaches as explained in Chapter 3, we have used them as baselines for comparison with our proposed approach. In the next chapter, we will define the adversarial navigation problem formally and then explain our solution.

Chapter 5

A Novel Approach to Optimizing Adversarial Navigation with RL

5.1 Defining Adversarial Navigation

The adversarial pathfinding problem is defined on a graph $G = (V, E)$ where V is the set of nodes and E is the set of valid edges between the nodes. Edges are represented as $E = \{(v, v') : (v \ \& \ v' \in V)\}$. Each edge has an associated positive cost represented by $c(v, v') \in \mathbb{R}^+$. A path ν is a sequence of nodes $\nu = v_0, v_1, v_2, v_3 \ldots v_k$ such that $(v_i, v_{i+1}) \in E$ for every $i \in \{0, 1, 2, \ldots k-1\}$. The i-th node on the path is represented using ν^i. The cost of traversing a path is calculated by summing all the edges in ν:

$$cost(\nu) \;=\; \sum_{i=0}^{k-1} c(\nu^i, \nu^{i+1})$$

Enemies (ξ) are defined as units whose objective is to inflict damage on the agent. The adversarial path planning problem is modelled by considering enemies as part of the environment. Enemies have a defined set of actions, which in this research are: move to an adjacent vertex, stay at the same location, and attack the agent. An enemy ξ_i can cause damage of magnitude $\hat{A}_i$ to the agent. Graph G is an adversarial environment with N enemies located at vertices: $\hat{v_1}, \hat{v_2}, \hat{v_3}, \ldots \hat{v_N}$.

The agent starts with an initial health H at location v_s. It's destination is v_d. The agent has a legal set of actions, which in this research are moves to adjacent vertices. If ξ_i is successful in its attack on the agent, then the health of the agent after the attack

is: $H - \hat{A}_i$. The agent is killed if its health falls to 0 or lower. Therefore, we can define the input to the adversarial navigation problem using the tuple $\langle G, c, \xi, \hat{A}, H, v_s, v_d \rangle$. The behaviour of ξ is not known but assumed to be static. The objective of the agent is to reach the destination by simultaneously reducing the distance travelled and the damage it takes from the enemies. The trade-off occurs because minimizing distance requires the agent to follow specific shortest distance paths which can cause frequent enemy encounters and high damage. Minimizing enemy encounters could require the agent to plan longer paths that avoid areas with many enemies.

In our experiment, G is an 8-connected (octile) grid map, but the approach is extensible to other pathfinding architectures.

5.2 Our MDP For Adversarial Navigation

We design a Markov decision process (MDP) that represents the general adversarial navigation problem. In our game design, the world is deterministic and consists of fixed obstacles and dynamic enemies. The agent can only observe a limited area of the map around it, called the agent's field of view (FOV) (Figure 5.1). The agent's FOV is much smaller than the area of the entire map. This means that changes in the map are not completely predictable when the agent moves, as new obstacles come into the FOV. Similarly, enemies have their own strategies for movement and may appear or disappear from the FOV. As a result, the agent's view of the dynamics of the game are modeled by an MDP.

States in the MDP are defined by the agent's field of view. The image of the FOV defines half of the MDP state. An additional set of features are computed using information from the agent's FOV as the other half of the MDP state. These derived features contain the locations of nearby enemies, and will be discussed in the Q-Network architecture section.

In this work our enemies are identical and have fixed strategies for chasing and attacking the agent. Similar to the agent, the enemies have visibility of the portion

of the map that is within their FOV. An enemy will chase and attack the agent if the agent is in their FOV. To execute a chase, the enemies predict the agent's next location by tracking the agent's previous location, and planning a path towards the agent's predicted next location. Enemies attack the agent by collision. Further details of the game design are provided in the section on experimental setup.

Actions in the MDP are defined relative to the direction of the global PRA* path at its point closest to the agent. The legal actions include going straight and turning 45 or 90 degrees in either direction relative to the global path.

The agent is trained on the MDP using reinforcement learning, so the learned policy depends on the rewards in the MDP. We have two objectives for agents in the game: (1) they should take short paths from the source to the destination and (2) they should avoid taking damage from enemies on the way. The rewards in the MDP are used to balance these objectives during learning.

5.3 Encoding Objectives in the Rewards Function

The states of the MDP are split into two sets with two different reward structures. The first set of rewards applies when the agent is safe with no enemies in the vicinity, and the second set when the agent is unsafe due to presence of enemies.

An agent's state at time t is represented using λ_t^U if the agent is in an unsafe state and λ_t^S if the agent's state is safe. In safe states the rewards are designed such that the agent learns to follow the global path. If the agent is at node ν^i on the global path then any action that moves the agent to node $\nu^j : j > i$ receives a positive reward *RewardAdvance* (all reward values are provided in Table 7.1). Safe states on the global path are terminal states of the MDP. That is, an episode ends when the agent reaches a location on the global path that is ahead of its previous location on the global path. Thus the agent's trajectory during gameplay from v_s to v_d consists of, from a learning perspective, many, potentially short, episodes. This simplifies the learning process and allows agents to start learning quickly.

Other safe states that are not on the global path, but have the global path in the agent's field of view, are non-terminal states of the MDP. If an agent's action transitions the agent to a non-terminal state at unit distance from the global path, the agent receives a reward *RewardNonTerminal_1_Safe*. Similarly, if the agent's action results in transition to a non-terminal state located 2 units from the global path, the reward is *RewardNonTerminal_2_Safe*. In the same manner, we have defined rewards (*RewardNonTerminal_3_Safe*) for actions that transition the agent to states that are at 3 or more units away from the global.

Safe states that do not have the global path in the agent's field of view are terminal states and represent the case that the agent is unrecoverable or lost. An action that transitions the agent to a lost state is rewarded negatively with *RewardLost*. This is also a terminal state. The agent starts a new episode by resuming at the same location with a new global path from PRA*.

In unsafe states the rewards are designed such that the agent learns to reduce damage and reach a safe state. Consider three possible sequences of states (ψ_1, ψ_2, and ψ_3) the agent may encounter while moving from v_s to v_d. The hyphens in the sequences are used indicate when separate episodes begin. A safe state on the global path is a terminal state and therefore the agent's trajectory can contain many episodes.

$$\psi_1 = \lambda_0^S - \lambda_1^S \lambda_2^S \lambda_3^U \lambda_4^U \lambda_5^U \lambda_6^U \lambda_7^S - \lambda_8^S$$

$$\psi_2 = \lambda_0^S - \lambda_1^S \lambda_2^S \lambda_3^U \lambda_4^U \lambda_5^S - \lambda_6^S - \lambda_7^S \lambda_8^S$$

$$\psi_3 = \lambda_0^S \lambda_1^S - \lambda_2^S \lambda_3^S \lambda_4^S - \lambda_5^S \lambda_6^S \lambda_7^S \lambda_8^S$$

The adversarial navigation trajectory represented by ψ_3 is simpler because it only requires making progress on the global path through safe states. Since ψ_1 has more unsafe states than ψ_2, we consider ψ_1 to be further from the solution to the problem of adversarial navigation than ψ_2. Therefore we designed our rewards such that the agent learns to avoid damage in the unsafe states while looking to move to a safe

state on the global path. This reduces both path cost and damage from enemies. In practice, once the agent has learnt to navigate within safe states, it will learn to navigate unsafe states with strategies that neutralize enemies the fastest. For example, in a certain unsafe state, the agent can decide whether an escape strategy (defensive) is better than an attacking (offensive) strategy based on which strategy can defeat enemies more quickly.

We hypothesize ($\mathcal{H}$) that the design of our rewards will lead to an improved solution for adversarial navigation. The hypothesis $\mathcal{H}$ is validated if the proposed approach results in reduced damage and shorter paths than the baseline approaches. We will validate the hypothesis empirically through experiments.

The agent is provided a negative reward *RewardNonTerminalUnsafe* when its current state is an unsafe state, and its next action leads to another unsafe state. If an agent-action results in damage to the agent due to an attack from a nearby enemy, then it receives a large negative reward *RewardInjury*. Furthermore, an agent action that transitions the agent to a lost state also receives a negative reward *RewardLost*. Both agent states where the agent is either damaged or lost are terminal states. The agent starts a new episode by resuming at the same location where the previous episode terminated. The learning follows the reward structure of safe state to safe state transitions when the agent's action transitions it from an unsafe to a safe state.

5.4 Q-Network Architecture

This section describes the architecture of the learning framework, including the inputs and outputs of each module (Figure 5.1).

Features are extracted from the agent's field of view (FOV) through a combination of 2D convolution layers and environmental representations (refer to Chapter 6). Environmental representations are computed features that generalize well across different environments. For example, the agent uses raycasts to spot the position of obstacles around it, abstracting actual shapes and sizes of the obstacles. Furthermore,

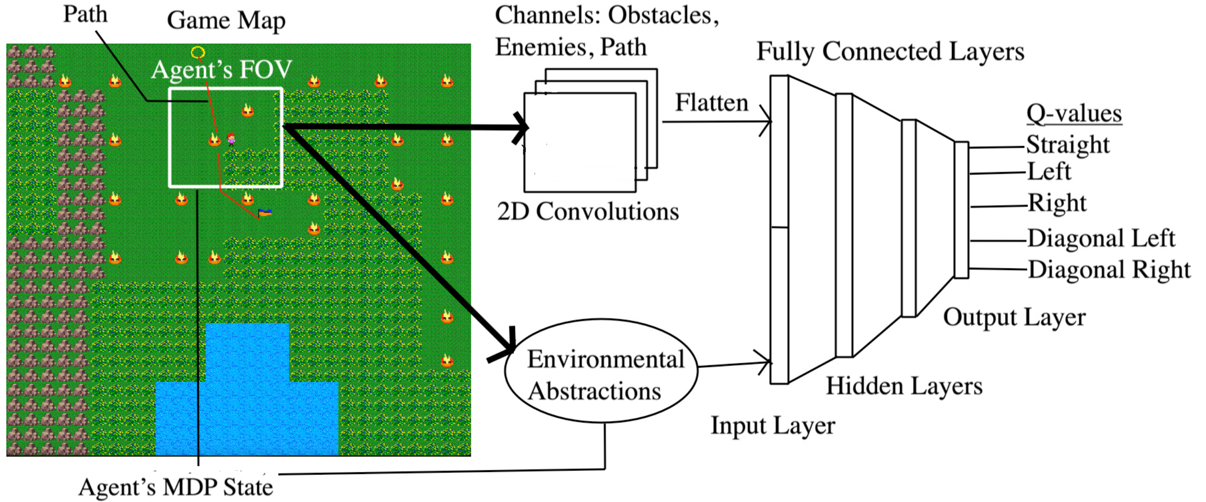

Figure 5.1: A model of the neural network for learning navigation in adversarial environments.

instead of tracking the entire section of the path inside the FOV, the agent abstracts it by finding the point on the path that is closest to the agent and the direction of the path at that point. The agent evaluates linear and angular distances to enemies in the FOV relative to its current direction of motion. These measures are invariant to FOV rotations which happen when the agent's direction changes as it navigates. Based on the linear and angular distances to an enemy, a risk score is assigned to it. Risk scores are higher for enemies located close to the agent or having frontal placement, while risk scores are lower for enemies further away or located in the rear of the agent. The enemies are then sorted by their risk scores. This helps the agent prioritize threats and also reduce variance in the training data. Environmental representations are designed to reduce the size and variance of the feature space through extraction and processing of key information while abstracting unnecessary details. They also help generalize the learning to unseen environments (i.e. environments not used in training the agent). A detailed explanation on the environmental representations are provided in the Chapter 6.

We observed that complementing the environmental representations with a single 2D convolution layer improves model accuracy without compromising on generaliz-

ability or speed of training. The dimensions of the agent's FOV is 9×9. The CNN has 3 input channels that capture the position of obstacles, enemies, and the global path in the FOV. The kernel size is 3, and there are 16 output channels of the CNN. The output of the CNN is then passed through a *ReLU* activation function before being flattened into a one dimensional tensor.

The agent computes abstract features through the environmental representations module and creates a single dimensional tensor. The two tensors are concatenated forming the agent's complete state tensor. The state tensor of size 834 is the input layer of a fully-connected neural network with 2 hidden layers and 1 output layer. The first hidden layer contains 520 neurons, and the second hidden layer contains 400 neurons. The output of the hidden layers pass through a *ReLU* activation before entering the next layer. The output layer contains the same number of neurons as the agent's action space. In this research, the agent's action space consists of 5 actions determined relative to the agent's current direction: move straight, move left, move right, move diagonal left, move diagonal right.

Given an agent state, the output layer produces Q-values associated with every action from that state. The neural network is trained by minimizing the loss function, as explained in Section 4.1.1. During training, the agent follows an ϵ-greedy strategy - choosing the action having highest Q-value with a probability of $1-\epsilon$ (exploitation) and a random action (exploration) otherwise. The value of ϵ is initially set to 1 to encourage exploration over exploitation and then decayed gradually until the value of 0.25 is reached after which no more decay is allowed. The loss function is optimized with stochastic gradient descent using the Adam optimizer and gradient clipping. A replay buffer of capacity 15000 is used to store past experiences, from which a batch of 4000 is randomly sampled in every epoch of training. Training is run for 300,000 epochs.

In this chapter, we defined the problem of adversarial navigation formally, then provided a detailed description of our approach based on reinforcement learning. We

also explained the architecture of the neural network used to train the agent. We also introduced environmental representations which are a set of computed features on which the neural network is trained. The next chapter is a deeper dive into these features.

Chapter 6

Environmental Representations

In this chapter we will discuss the environmental representations, that are used as part of the Q-Network architecture (Section 5.4), in detail. We assume that the world is partially observable to the agent. However, the agent has complete knowledge of the portion of the world which is within its field of view (FOV). The agent's FOV defines the agent's current state. Any change to the FOV is a different state for the agent. The agent's FOV also includes enemies that are located within the agent's FOV. Additionally, the same FOV but observed from different directions, such as the agent facing North vs. the agent facing East, are different states for the agent. The FOV of the agent can vary extensively within a single environment and also between different environments. As a result, learning a representation of the FOV is challenging due to the possible number of features and variance in the date. Furthermore, this process of learning a representation of the FOV from all possible features can prevent the training from converging due to the large state space of the FOV, or cause overfitting due to learning from features that do not generalize to unseen environments. Therefore we developed certain abstract features which we call environmental representations with the objective of capturing only essential information while leaving out superfluous details. The environmental representations are manually crafted using domain knowledge that can generalize across environments.

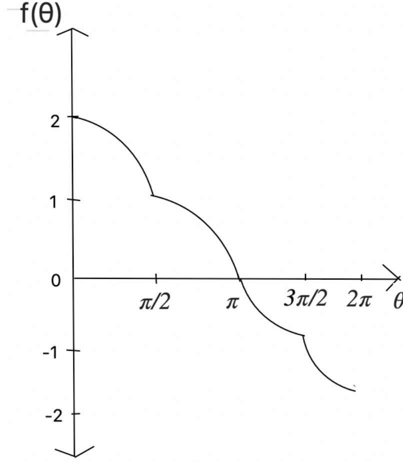

Figure 6.1: Plot of the normalizing function $f(\theta)$

6.1 Risk Representation

Each enemy within the agent's FOV is assigned a risk score based on its Manhattan and angular distance from the agent. The closer an enemy is to the agent, the greater its risk score. Similarly, enemies that are located towards the front of the agent are more capable of inflicting damage to the agent than enemies located towards the rear. To deduce an expression for the risk score, we first define a normalizing function $f(\theta)$ where θ is the enemy's angular distance from the agent's direction of motion.

Table 6.1: Function to normalize angular distance of enemies

$f(\theta) = \cos\theta + 1$, in quadrant 1	$f(\theta) = \sin\theta$, in quadrant 2
$f(\theta) = \sin\theta$, in quadrant 3	$f(\theta) =$ -$(\cos\theta + 1)$, in quadrant 4

$f(\theta)$ is calculated according to Table 6.1 and shown in Figure 6.1. $f(\theta)$ is positive in the interval $[0, \pi]$, which is the left side of the agent, assuming angles are measured counter-clockwise. It is negative in the interval $[\pi, 2\pi]$, which is the right side of the agent. The absolute value of $f(\theta)$ approaches 2 for enemies positioned to the front of the agent and nears 0 for enemies located near the rear of the agent.

The risk score η for an enemy with a Manhattan distance d and angular distance θ is calculated below:

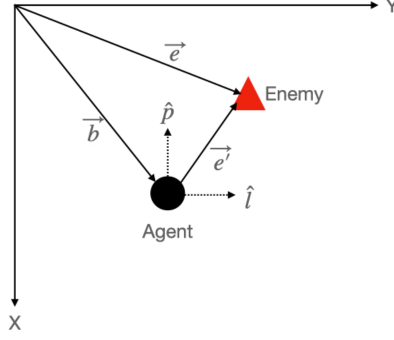

Figure 6.2: Change of coordinate system for rotation abstraction.

$$\eta = \left| f(\theta) \right| e^{-0.9d} \tag{6.1}$$

Risk scores of enemies are included in the agent's state space. Enemies within the agent's FOV are also sorted by risk scores. Sorting reduces the agent's state space since ordering of features reduces many states to a single state. The top five risk scores are added to the input tensor of the neural network (Figure 5.1).

6.2 Rotation Abstraction

As the agent traverses the global path, it changes direction at the turns. The agent's FOV is computed relative to the agent's direction. As a result, the FOV around the agent rotates either clockwise or anti-clockwise. This abstraction ensures that the agent does not consider a rotated FOV as a new state. In this section, we derive expressions that enable the agent to be invariant to FOV rotations when the agent's direction change.

The world is represented using a 2D grid world with (x, y) coordinate system, as shown in Figure 6.2. The agent's position is represented with $\vec{b}$ and enemy's position is $\vec{e}$, both of which are relative to the origin. $\hat{l}$ is a unit vector along the agent's direction of movement and $\hat{p}$ is a unit vector orthogonal to $\hat{l}$. The enemy's position relative to the agent's coordinate system, represented by $(\hat{l},\, \hat{p})$, is $\vec{e'}$. Unlike $\vec{e}$, $\vec{e'}$ is rotation invariant. Therefore, the objective is to deduce the location of the enemy

$(\vec{e'})$ relative to the agent's direction of motion given the agent's coordinates $(\vec{b})$, the enemy's coordinates $(\vec{e})$ and the agent's direction of motion.

Using vector arithmetic, we have (Figure 6.2):

$$\vec{b} + \vec{e'} = \vec{e} \tag{6.2}$$

We now split $\vec{e}$ into its x-component e_1 and y-component e_2 and the agent vector into its x-component b_1 and y-component b_2.

$$\vec{e} = e_1\hat{x} + e_2\hat{y} \tag{6.3}$$

$$\vec{b} = b_1\hat{x} + b_2\hat{y} \tag{6.4}$$

The vectors $\hat{l}$ and $\hat{p}$ can be represented similarly.

$$\hat{l} = l_1\hat{x} + l_2\hat{y} \tag{6.5}$$

$$\hat{p} = p_1\hat{x} + p_2\hat{y} \tag{6.6}$$

We can represent $\vec{e'}$ relative to the agent's coordinate system $(\hat{l}, \hat{p})$ using the components (e'_1, e'_2)

$$\vec{e'} = e'_1\hat{l} + e'_2\hat{p} \tag{6.7}$$

Substituting 6.5 and 6.6 in 6.7.

$$\vec{e'} = e'_1(l_1\hat{x} + l_2\hat{y}) + e'_2(p_1\hat{x} + p_2\hat{y}) \tag{6.8}$$

Substituting 6.8, 6.4 and 6.3 and in 6.2.

$$b_1\hat{x} + b_2\hat{y} + e'_1(l_1\hat{x} + l_2\hat{y}) + e'_2(p_1\hat{x} + p_2\hat{y}) = e_1\hat{x} + e_2\hat{y} \tag{6.9}$$

Rearranging terms in 6.9

$$(b_1 + e'_1 l_1 + e'_2 p_1)\hat{x} + (b_2 + e'_1 l_2 + e'_2 p_2)\hat{y} = e_1\hat{x} + e_2\hat{y} \tag{6.10}$$

Equating $\hat{x}$ on both sides of equality of 6.10, we get the following:

$$e'_1 l_1 + e'_2 p_1 = e_1 - b_1 \tag{6.11}$$

Equating $\hat{y}$ on both sides of equality of 6.10, we get the following:

$$e'_1 l_2 + e'_2 p_2 = e_2 - b_2 \tag{6.12}$$

From 6.11 and 6.12 we can solve for e'_1 and e'_2. This then tells us $\vec{e'}$, which represents the enemy's position relative to the agent's direction of motion. $\vec{e'}$ is invariant to FOV rotations due to change in agent's direction. The positioning relative to the agent's coordinate system $(\hat{l}, \hat{p})$ abstracts the agent's direction with respect to the actual coordinate system $(\hat{x}, \hat{y})$.

We solve 6.11 and 6.12 using matrix operations.

$$\begin{bmatrix} l_1 & p_1 \\ l_2 & p_2 \end{bmatrix} \begin{bmatrix} e'_1 \\ e'_2 \end{bmatrix} = \begin{bmatrix} e_1 - b_1 \\ e_2 - b_2 \end{bmatrix} \tag{6.13}$$

Solving for e'_1 and e'_2, we get:

$$e'_1 = \frac{p_2 e_1 - p_2 b_1 - p_1 e_2 + p_1 b_2}{l_1 p_2 - p_1 l_2} \tag{6.14}$$

$$e'_2 = \frac{l_2 b_1 - l_2 e_1 + l_1 e_2 - l_1 b_2}{l_1 p_2 - p_1 l_2} \tag{6.15}$$

The distance of the enemy from the agent is given by $|\vec{e'}| = \sqrt{(e'_1)^2 + (e'_2)^2}$. The enemy's angular distance relative to the agent's direction is $\theta = \arccos \frac{e'_1}{|\vec{e'}|}$.

Linear and angular distances of the enemies with the top five risk scores are calculated. These values are appended to the input tensor of the neural network (5.1).

6.3 Simple Grid-based Raycast Distances

The agent uses raycasts to determine the position of obstacles around it (Sauze and Neal, 2010). It does so by scanning for obstacles in a radially outward manner. The information obtained as a result abstract away details of the environment that are not necessary to build effective navigation strategies. For example, if a unit cannot swim, it would consider both water bodies and impenetrable forests as obstacles. Furthermore, the agent does not need to know the exact shape and size of the obstacles to decide which moves are blocked by obstacles. Raycasts precisely conveys the information about obstacle distances while abstracting unnecessary details.

The agent also uses raycasts to locate the global path relative to its direction of motion. Once a node on the path is identified, the agent determines the direction of the path at that node by analyzing the adjacent nodes that lie on the path. Raycast distances are computed by scanning (outwards) for obstacles in the direction of the agent's actions: straight, 45 degrees to the left, 90 degrees to the left, 45 degrees to the right, and 90 degrees to the right. As a result of the scan, the distance of obstacles in the direction of raycasts are included in the input tensor of the neural network (5.1).

6.4 Path Representation

Within the agent's FOV, the characteristics of the global path can vary widely. A model that considers detailed path characteristics while making a decision is likely to overfit and generalize poorly. Therefore, in the proposed approach, the agent extracts relevant abstract features from the path while discarding unnecessary details. The agent first uses raycasts to locate the path. Then, it scans neighbouring nodes on the path within the FOV to decide direction. The agent chooses the direction having maximum overlap with the path. For example, in Figure 6.3, the agent's directions are North in position 1, East in positions 2 and 5, and South in positions 3 and 4.

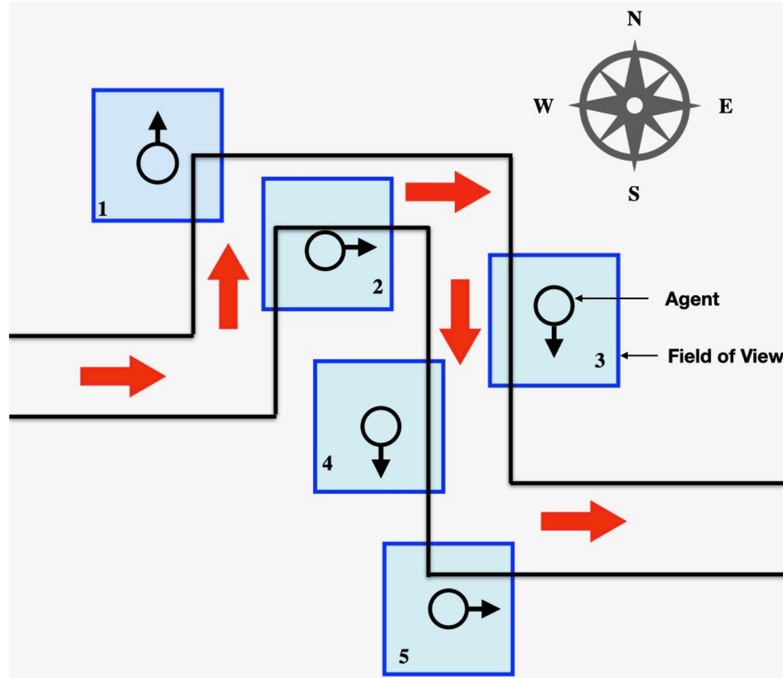

Figure 6.3: Possible positions of the agent along the path is shown. Path directions are marked with red arrows. Agent's direction is inferred and shown with black arrows.

The abstraction implies that the agent only considers the path location and path direction for its decision, ignoring detailed path characteristics like the actual shape of the path inside the agent's FOV.

The agent combines the path representation abstraction with rotation abstractions (Section 6.2) to prune states. For example, in Figure 6.3, positions 1 and 3 are identical to the agent as well as positions 2, 4, and 5. In positions 1 and 3 the agent determines that the path is to its right by considering the amount of overlap of the path with the FOV and the path's direction in the maximum overlapped portion of the path. In a similar manner, for positions 2, 4 and 5 the the agent determines that the path is to its left. The agent ignores other path characteristics such as the exact shape and nature of the path because consideration of detailed path characteristics is unnecessary for the agent's immediate decisioning. Thus, the agent can compress 5 different states into only 2 distinct states. This is another way the agent reduces variance in the training data and the size of the state space of the MDP.

Using path representations the agent determines its direction and the relative position of the global path. The global path can be located to the agent's left, right or in-front. The agent can also be located on the global path. This information is finally provided to the neural network by appending four binary values [on path, left, right, in-front] to the input tensor. For example, if the agent is on the path, then these values would be: 1, 0, 0, 0.

Chapter 7

Experimental Results

In this chapter, we discuss in detail the experiments we have conducted to test whether adversarial navigation can be approximately solved through our approach. This chapter is organized as a list of experiments. For each experiment we provide its objective, experimental setup, observations, and a discussion of results.

Figure 7.1: A generated map used to train the agent.

For all of our experiments we have designed a simplified adversarial game, as shown in Figure 7.1. The agent is represented as a boy. The objective of the agent is to navigate from the start position (marked with a circle) to the goal position (marked with a flag). The map is populated with fire monsters, which are enemies capable of inflicting damage to the agent. If the agent is in the field of view of the fire monsters,

they chase the agent. A collision with a monster would cause 10 points of damage to the agent. Each fire monster is territorial and stays within a radius of 6 units from its starting location. All other objects, such as rocks, water, trees, etc., are considered obstacles. The fire monsters have one weakness: If two or more monsters collide with each other, they are both destroyed and removed from the game. The agent is unaware of this weakness. However, we expect the agent to eventually discover this enemy weakness and learn to exploit it.

The agent is trained on 10 maps of size 27×27 with obstacles uniformly placed, as shown in Figure 7.1. These maps differ in the initial position of the fire monsters, which are calculated by random selection from a uniform distribution. At each iteration of training, the agent's start and goal locations are decided randomly. The agent's trajectory from the start to the goal is split into many training episodes. An episode ends if the agent can reach a location on the global path that is safe and ahead of agent's last location on the global path. Additionally, an episode ends when the agent is killed by an enemy or has lost the global path from its FOV. When an episode ends, the agent starts a new episode by resuming at the location the previous episode ended. This process continues until the agent reaches the destination or a timeout occurs. A timeout will occur if the agent has travelled 5 times the optimal path length and yet not reached the destination. If the agent has reached the destination or a timeout occurred, the agent will start a new episode of adversarial navigation with new start and goal location on the same game map or a different one. The agent is trained with reinforcement learning using the reward structure shown in Table 7.1.

The agent can request a new path if it considers proceeding on the current path to be too risky. The agent consider a state risky when Q values for all available actions in that state are lower than a threshold. The threshold is set to -22 and determined through manual tuning by trial and error. This threshold value has not been modified in any of the experiments.

State	Name	Value
Safe	RewardAdvance	15
Safe	RewardNonTerminal_1_Safe	−1
Safe	RewardNonTerminal_2_Safe	−2
Safe	RewardNonTerminal_3_Safe	−3.5
Unsafe	RewardInjury	−45
Unsafe	RewardNonTerminal_Unsafe	−3.5
Both	RewardLost	−45

Table 7.1: Rewards used to train the agent

There are two baseline agents for comparison. The first one is based on PRA* (Sturtevant and Buro, 2005). The PRA* agent computes an abstract path from the source to the destination and then refines up to 8 abstract nodes in the real world. While pathfinding in the real world, the agent considers enemies as virtual obstacles. It is possible to prevent this baseline agent from going close to the enemies (whenever possible) by treating enemies as larger obstacles. The second baseline is based on potential fields (Hagelbäck, 2016). The goal is given a low potential, and the enemies and obstacles are assigned high potential values. The agent moves in the direction of the steepest drop in potential.

This experiments compare the proposed approach against the baseline approaches on the following measures: 1) Amount of damage the agent takes, 2) Path length sub-optimality, 3) Maximum memory used for pathfinding, and 4) Execution time of episodes. For the purpose of measuring the amount of damage and comparing the approaches, the agent is given infinite life. At the end of an episode of adversarial navigation, the total damage incurred by the agent is recorded. Path length sub-optimality is calculated as $\frac{ActualPathLength}{OptimalPathLength}$. Since PRA* is based on the A* (Hart, Nilsson, and Raphael, 1968) algorithm, it requires an open and a closed list for path finding. The maximum size of the open list is a measure of the maximum mem-

ory used. The execution time represents the total time to complete an episode in milliseconds.

For convenience, we call the agent using the proposed approach the RL agent. The baseline agent using PRA* is named the PRA* agent. Similarly, we call the baseline agent using potential field the PF agent.

All experiments are conducted on a Macbook with Apple M1 processor and 16 GB RAM. C++ is used for programming and LibTorch is used for Machine Learning.

7.1 Experiment 1 : Adversarial Navigation on the Warcraft III Maps

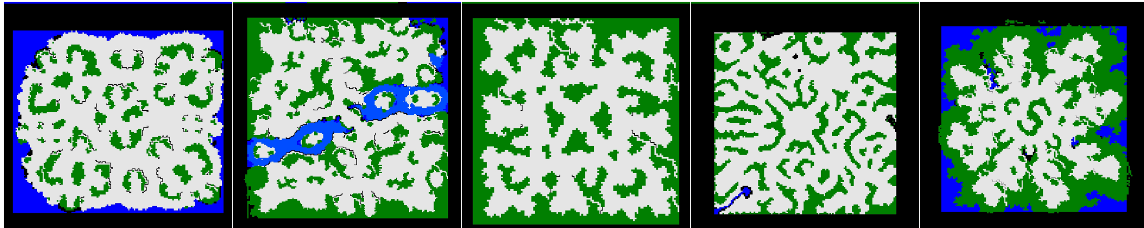

Figure 7.2: Warcraft 3 maps used to test the agent. From left to right: blastlands, divfideandconquer, duskwood, gardenofwar, and thecrucible.

This experiment compares the proposed approach with baseline approaches on the maps of the popular RTS game Warcraft III, developed and published by Blizzard Entertainment. The maps are scaled versions of the original maps and have dimensions 512×512 (Sturtevant, 2012). It is important to note that these maps have not been used for training the agent. Furthermore, the maps of Warcraft III (Figure 7.2) are significantly different from the maps used for training the agent (Figure 7.1). Therefore this experiment also checks the generalizability of our approach. Five Warcraft III game maps: blastedlands, divideandconquer, duskwood, gardenofwar, and thecrucible (Figure 7.2), were randomly selected. We then generated 2000 random paths per map evenly binned by path lengths. The bin size is 50. The number of enemies in each of the Warcraft III maps is provided in the Table 7.2. The enemies are

distributed in the maps randomly with the only constraint that two enemies would be placed at least 3 units apart.

We observed that the proposed approach takes 4 times less damage than the baseline approaches on average without compromising on path length. The proposed approach has similar memory efficiency to the baseline approaches and is comparatively faster in execution time than the baseline approaches. Furthermore, the proposed approach's superior performance on different types of test maps (Figure 7.2) in addition to the custom training maps (Figure 7.1) demonstrate its generalizability. The experimental results on the training maps is provided in Section 7.3.

Map	Total Enemies
Blastlands	6276
DivideAndConquer	5779
Duskwood	5984
GardenOfWar	5739
TheCrucible	3904

Table 7.2: Number of enemies in Warcraft III maps

The baseline approaches require hand-tuning of parameters for every game environment setup used in the experiments. This is often tedious and challenging. In the case of PRA* baseline, there is a need to set the size of the virtual obstacles. For the potential field based baseline the attraction and repulsion values need to be tuned. We plotted all the figures in this section with our best possible configuration for PRA* and potential fields for each experiment.

Since both the baseline agents have a similar evasion strategy, which is to stay away from the enemies, they accrue similar damage points for similar path lengths (Figure 7.3). However, the proposed approach (RL) takes significantly less damage, which implies that the learnt strategies are more effective than the simple enemy evasion strategy the baseline agents use. For example, the enemy evasion strategy is

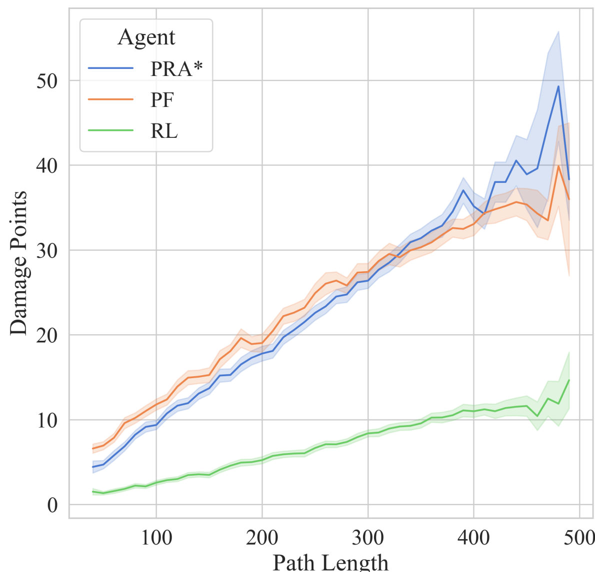

Figure 7.3: Amount of damage the agent takes, plotted against optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors

ineffective when the agent is encircled by adversaries. We observed that the RL agent is less vulnerable to encirclement as its strategies more aggressively exploit enemy weak points, allowing little time for enemy build-up around it.

The length of the actual paths navigated by the RL and the PRA* agents are more often closer to the optimal path than the PF agent (Figure 7.4). This is because both approaches explicitly minimize path cost. However, the PF agent is not penalized for taking longer paths. We also observed that the RL agent is less sub-optimal than PRA* in path length. This shows that the RL agent has learnt to counter enemies while staying close to the global path, as a result minimizing path length suboptimality. This affirms our hypothesis ($\mathcal{H}$) from Section 5.3.

Both the RL agent and the PRA* agent use additional memory for pathfinding. The PRA* paper defines a constrained A* search which significantly reduces the search space. This results in lower memory footprint and execution time compared to a regular A* search. However, we observed that in a dynamic environment refin-

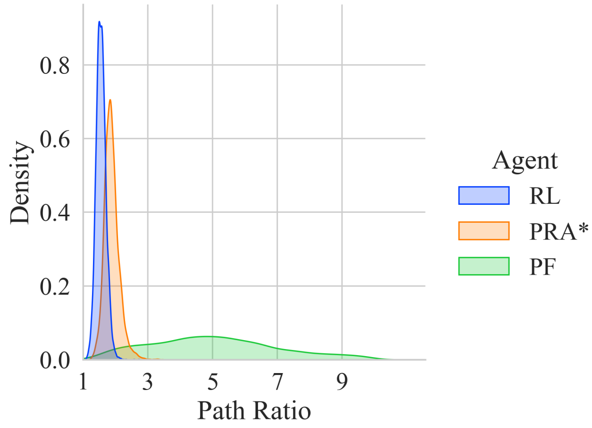

Figure 7.4: Path length suboptimality for different types of agents.

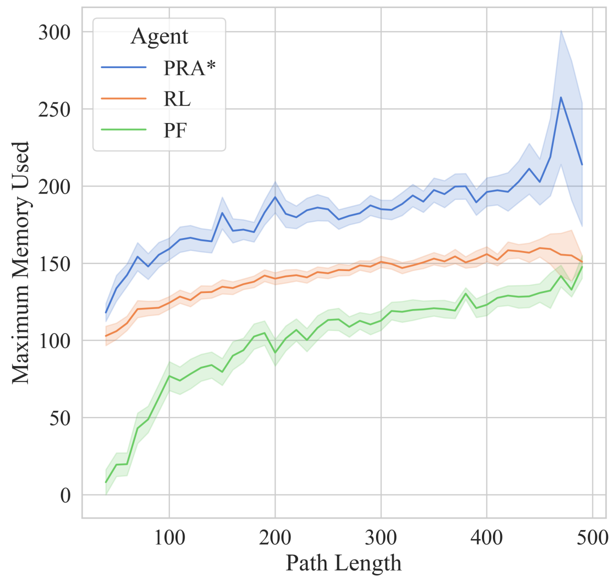

Figure 7.5: The maximum memory used for path finding measured by the size of the open list, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors

ing narrow corridors as part of the constrained A* search can be unsuccessful when enemies accumulate in such corridors blocking the path completely. One solution to this problem is to expand the size of the corridors. However, since this amounts to relaxing the constraint of the constrained A* search, it will increase memory use and execution time compared to the unrelaxed constrained A* search. In our experiment, we dropped constrained A* search from the PRA* baseline to resolve the problem of unsuccessful refinement due to enemy congestion. This is the reason for high memory usage of PRA* (Figure 7.5). The RL-agent does not suffer from the enemy-congestion problem despite using PRA* because it does not consider enemies (and dynamic obstacles) during the pathfinding step. It tackles enemies and dynamic obstacles locally during navigation. Therefore, combining RL and PRA* together in the proposed approach is symbiotic, resulting in effective navigation in hostile environments. The potential field approach does not use any additional memory except when it is stuck in local optima (Hagelbäck and Johansson, 2021). We implemented repulsive trails to minimize the problem but still needed PRA* to recover the agent in the few instances the agent got stuck. Thus, we see the PF-agent using some additional memory in Figure 7.5.

The execution time of PRA* is the highest (Figure 7.6) because: 1) its heuristic function does not account for environment dynamics and is therefore less accurate, 2) it does not benefit from the constrained A* search due to the problem of enemy congestion. Note that the PRA*-agent can benefit from constrained-A* search partially if the size of the corridors are expanded. The RL-agent fully benefits from the constrained A* search and is therefore significantly faster in computing paths than the PRA*-agent. Moreover, in the case of the RL agent, inference from neural networks is a constant time operation with respect to the path length. However, the PRA* agent repeatedly searches for new paths to avoid adversaries, which has an exponential worst case time complexity in path length (Korf and Reid, 1998). Therefore execution time grows faster for the PRA* agent compared to RL agent.

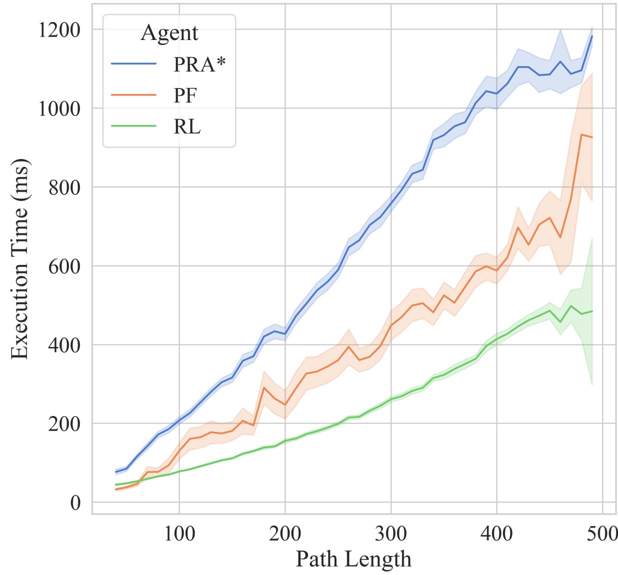

Figure 7.6: The amount of time in milliseconds that the agent took to reach the destination, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors

Agent	Damage		Path Ratio	
Statistics	Mean	Confidence Intervals	Mean	Confidence Intervals
RL	**6.55**	±0.1	**1.54**	±0.01
PRA*	21.94	±0.27	1.92	±0.04
PF	22.89	±0.25	6.13	±0.12

Table 7.3: Mean and 95% confidence intervals of damage and path ratio in experiment 1.

Time complexity of potential field calculation is linear in the branching factor of the map. It involves checking the potential values of the grid cells adjacent to the agent's current cell. Since in our octile grid maps the branching factor is constant, we can treat potential field calculation as a constant time operation in path length. Therefore, the execution time growth rate of the PF agent is slower than the PRA* agent. However, the PF agent is significantly more suboptimal in path length compared to

the RL agent and therefore requires a longer time to reach the destination.

The mean and 95% confidence intervals of path ratio and damage for all the agents are shown in Table 7.3.

7.2 Experiment 2 : Adversarial Navigation on the Warcraft III Maps After a Game Design Update.

Experiment 2 is also conducted using the five Warcraft III maps (Figure 7.2) described in the previous experiment. Additionally the number of enemies and the initial positions of the enemies in Experiment 2 are the same as Experiment 1. We updated the design of the game in Experiment 2. The objective of this experiment is to find out whether the agent can learn effective alternate strategies in a different game environment without any updates to the learning framework. If true, then this would imply that the proposed approach is generalizable across game designs. The results of this experiment is important because game designing is an iterative process. Our approach ensures that the agent can autonomously update its navigation strategies without code changes, when the game design is updated. Also, by not changing code of any agent we ensure that we do not accidentally give an advantage to the RL agent.

In the updated game design, we eliminated the vulnerability of the fire monsters. Instead, the fire monsters on collision fuse to become a single stronger enemy with greater attack points and range. The starting location of the fused enemies is the fusion site, which gives it an advantage since the fusion site is often located close to the agent's location. The design update resulted in the game becoming significantly more challenging than the first experiment for all types of agents (RL, PRA* and PF).

We observed that the agent was able to learn new strategies that worked well on the updated game design, even though no changes were made to the agent's learning

framework. The results of the second experiment are similar to the first experiment (in relative trends), with some differences in the magnitude of values (damage, path length sub-optimality, memory use, and execution time). The values are larger in the second experiment than the first because of stronger enemies in the second. It is also important to note that the strategies the agent has learnt in the first experiment are ineffective in the second experiment. Therefore, the RL agent must learn new strategies specific to the game's design in the second experiment for good performance.

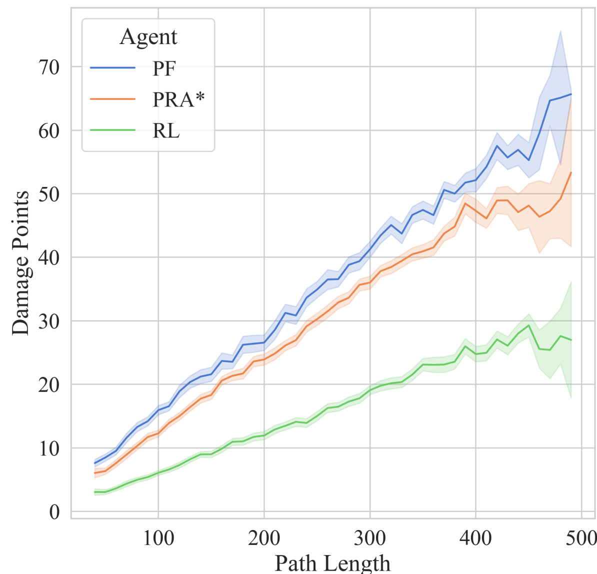

Figure 7.7: Amount of damage the agent takes in experiment 2, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors

Notice, Figure 7.7 is very similar to Figure 7.3, except that in this case all types of agents (RL, PRA*, and PF) take more damage. This happens because, in Experiment 2, the adversaries are stronger than in Experiment 1. The RL agent continues to dominate this metric with significantly less damage compared to the baseline agents. This confirms that the RL agent is able to learn alternate strategies that are effective in the updated game despite no changes in the learning framework.

Figure 7.8 compares path length suboptimality of different types of agents. The

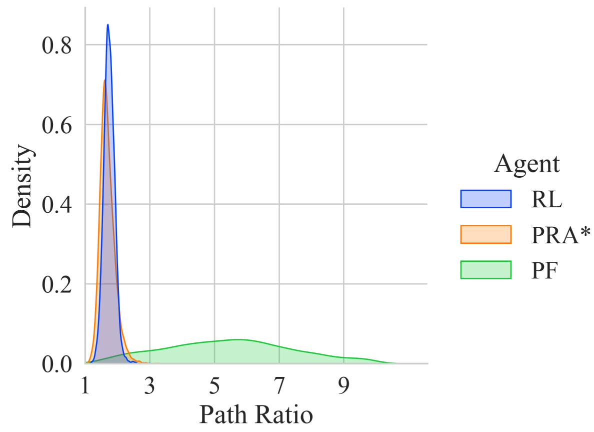

Figure 7.8: Path length suboptimality for different types of agents in experiment 2.

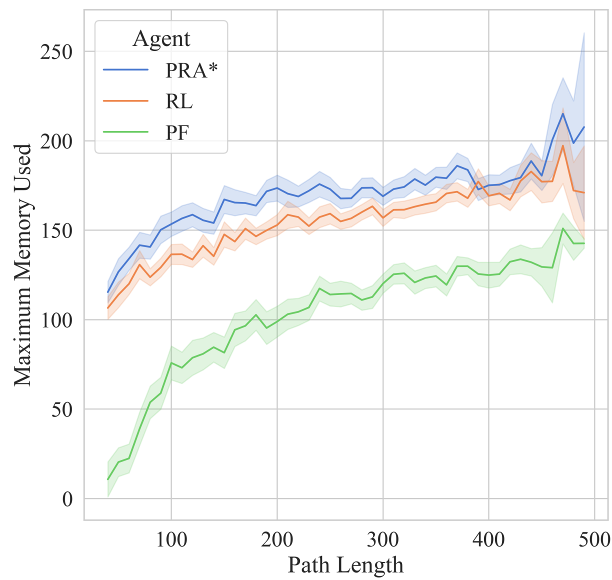

Figure 7.9: The maximum memory used for path finding measured by the size of the open list in experiment 2, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors

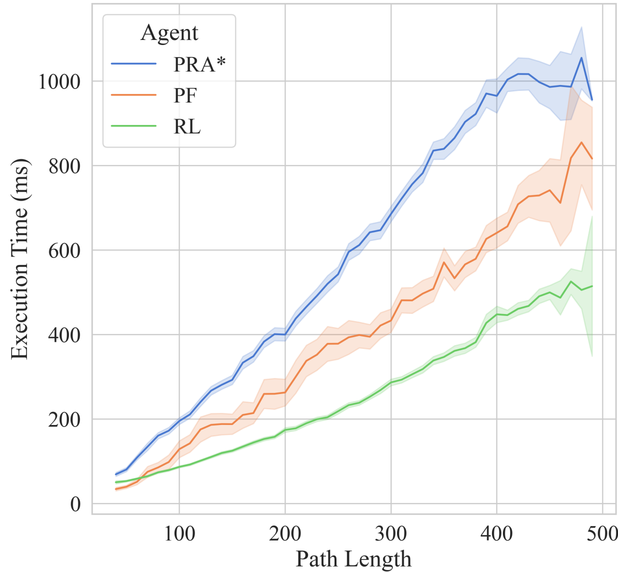

Figure 7.10: The amount of time in milliseconds that the agent took to reach the destination in experiment 2, plotted against the optimal path length. The paths are binned by path length with bin size 10. For each agent, we show the mean and two standard errors

results of Experiment 2 match the results of Experiment 1 (Figure 7.4). The RL agent is optimizing path cost and damage simultaneously better than the baseline agents in the updated game.

Next, we compare the different types of agents based on the amount of memory used for pathfinding and navigation. Figure 7.9 shows that the RL agent in Experiment 2 has consumed more memory than in Experiment 1 (Figure 7.5). This is due to the RL agent's re-pathing behaviour. Since the game episodes of Experiment 2 is more challenging than Experiment 1, the RL-agent requested new paths more often in Experiment 2 than in Experiment 1. Thus, we notice higher memory consumption of the RL agent in Experiment 2 than Experiment 1. However, the additional memory used by the RL agent still positions it in between the PRA* agent and the PF agent as in Experiment 1.

The behaviour of the agents with respect to execution time in Experiment 2 (Fig-

ure 7.10) is very similar to Experiment 1 (Figure 7.6) and hence needs no further explanation.

The mean and 95% confidence intervals of path ratio and damage for all the agents are shown in Table 7.4.

Agent	Damage		Path Ratio	
Statistics	Mean	Confidence Intervals	Mean	Confidence Intervals
RL	**15.12**	±0.2	**1.73**	±0.01
PRA*	29.09	±0.33	**1.72**	±0.04
PF	33.57	±0.38	6.6	±0.11

Table 7.4: Mean and 95% confidence intervals of damage and path ratio in experiment 2 having updated game design

7.3 Experiment 3 : Adversarial Navigation on the Training Maps

This experiment compares the proposed approach with baseline approaches on the training maps. We have shown in the the previous experiments that our approach can generalize to test maps that were not used for training. The purpose of this experiment is to verify the generalizability of our approach by reporting results on the training maps that are significantly different from the test maps.

There are 10 training maps of size 27×27. Due to the small size of the training maps, we generated 2000 paths of fixed length. The start and end locations were randomly picked such that the optimal path length between them is 26. For comparing the results, we have used a Raincloud plot (Allen et al., 2019), which shows summary statistics such as median and mean, and probability densities in an appealing format with minimum redundancy.

The RL agent takes less damage than the baseline approaches with no damage at all in more than 50% of the cases (Figure 7.11). The PRA* agent and the PF agent

Figure 7.11: Amount of damage the agents take in experiment 3.

performed similar to each other in the amount of damage they take from enemies. The results are similar to our observations in the other experiments.

Similarly, the RL agent on the training maps has lower suboptimality in the path length than the baseline approaches (Figure 7.12). The results are consistent with our observations from the experiments described earlier.

Furthermore, we found no difference in the results on maximum memory used (Figure 7.13) in comparison to the other experiments. PRA* agent used the maximum memory, followed by the RL agent and finally the PF agent.

In terms of execution time (Figure 7.14), we noted a difference from the other experiments. In this experiment, the RL agent is slower than the baseline agents. However, this result is not surprising since inference from Neural Networks, although constant time, is still an expensive operation. The Neural Network inference cost gets more pronounced because path finding in small maps is computationally cheap.

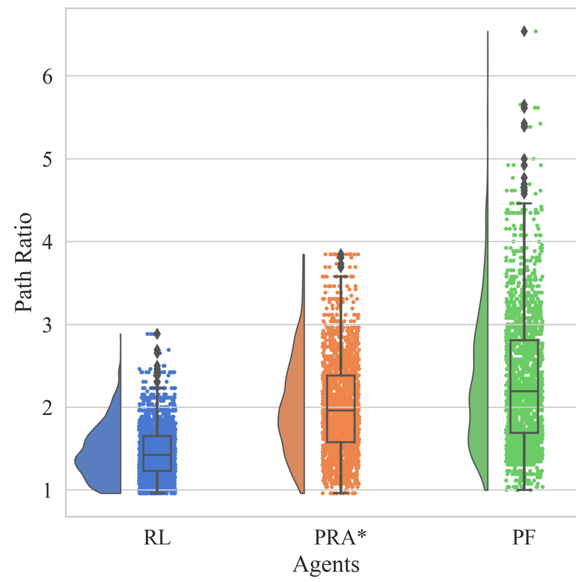

Figure 7.12: Path length sub-optimality for different types of agents in experiment 3.

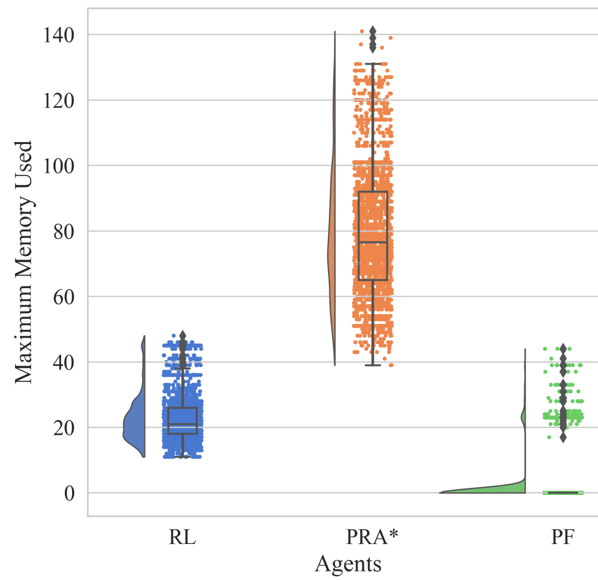

Figure 7.13: The maximum memory used by the agents for navigation in experiment 3.

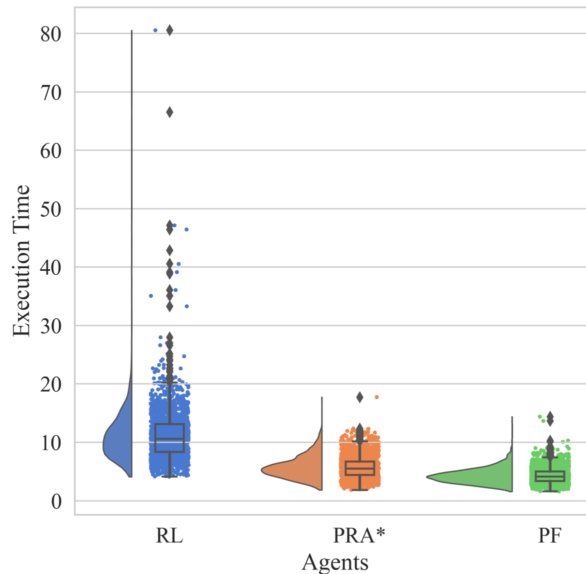

Figure 7.14: The amount of time in milliseconds that the agents took to reach the destination in experiment 3.

7.4 Limitations of the Proposed Approach

The RL agent's generalizability is limited by its state and action representation. As an example, the models we trained for the experiments described in this thesis are unlikely to work well in a shooting game. In a shooting game, the agent requires information such as ammunition left, types of weapons and their ranges, enemy weapons, etc. This information should be encoded in the agent's state. Furthermore, the agent's action space should be updated to contain actions such as select weapon, reload weapon, take aim at target, and shoot. The agent can then be trained to perform adversarial navigation in shooting games. The learning will generalize to other shooting games which were not used for training the agent.

We have also not explored the effectiveness of the approach in a full-scale RTS game such as StarCraft. There are two primary reasons for this. First, in a full-scale RTS game it is not easy to compare pathfinding approaches due to the influence of other objectives such as resource gathering, territorial expansion, trades, conquests, etc.

that are outside the scope of the current work. Secondly, swapping out the pathfinding and navigation algorithms from a full-scale RTS game is a major challenge.

The proposed technique for adversarial navigation has demonstrated success with a single type of adversaries having a defined behaviour. The research did not explore the case of adversaries with different behaviours. Additionally, the research's applicability to areas like MAPF and cooperative pathfinding also remains unanswered.

Finally, the research has used manually crafted features called environmental representations. While these features are explainable and reduce the complexity of the CNN, they are less portable in terms of implementation. As an example, environmental representations for a shooting game like Counter Strike could have differences from an adventure game like Prince of Persia which involves sword fights.

Chapter 8

Conclusion

The topic of adversarial navigation presents multiple objectives that must be simultaneously optimized to achieve the desired result. Previous approaches based on machine learning have failed to show that they generalize to unseen environments or scale to game maps of different sizes. Other popular approaches that are not learning based require manual tuning of parameters for different types of maps and adversaries. Through this thesis, we formalized the problem of adversarial navigation as an optimization task. We showed that reinforcement learning combined with PRA* can minimize path length and damage from adversaries simultaneously, without compromising memory or CPU time. We trained the agent on custom maps of size 27×27 and showed that the agent can outperform baseline approaches on maps of size 512×512 of the RTS game Warcraft III. This demonstrates our approach's generalizability, and ability to scale to large maps. In a second experiment, we updated the game dynamics to answer the question whether our approach can generalize across game designs. This property is useful since game designing is usually an iterative process. We observed that the agent could discover new strategies different from the ones used in the earlier game design. With the new strategies, the RL agent outperformed the baseline agents in the updated game.

The generalizability of the proposed approach is however limited by the agent's state and action space. For example, games of different genres would likely require

the agent to be trained on different feature sets, although, the core ideas for adversarial navigation discussed in this research stays intact. The proposed approach also used manually designed features called environmental representations. These features are less portable in terms of implementation than standard machine learning algorithms. However, these features are highly explainable and makes the learning more robust and generalizable. It is possible to swap out the environmental representations with appropriate machine learning approaches and larger models, albeit at the cost of explainability and greater training time and computation resources.

Bibliography

Adaixo, Michaël Carlos Gonçalves (2014). "Influence Map-Based Pathfinding Algorithms in Video Games". PhD thesis. Universidade da Beira Interior (Portugal).

Allen, Micah, Davide Poggiali, Kirstie Whitaker, Tom Rhys Marshall, and Rogier A Kievit (2019). "Raincloud plots: a multi-platform tool for robust data visualization". In: *Wellcome open research* 4.

Alonso, Eloi, Maxim Peter, David Goumard, and Joshua Romoff (2020). *Deep Reinforcement Learning for Navigation in AAA Video Games*. arXiv: 2011.04764 [cs.LG].

Anhalt, James (Mar. 2011). *AI Navigation: It's Not a Solved Problem - Yet*. GDC AI Summit. URL: https://www.gdcvault.com/play/1014514/AI-Navigation-It-s-Not.

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath (2017). "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6, pp. 26–38. DOI: 10.1109/MSP.2017.2743240.

Avery, Phillipa, Sushil Louis, and Benjamin Avery (2009). "Evolving coordinated spatial tactics for autonomous entities using influence maps". In: *2009 IEEE Symposium on Computational Intelligence and Games*, pp. 341–348. DOI: 10.1109/CIG.2009.5286457.

Avontuur, Tetske, Pieter Spronck, and Menno Van Zaanen (2013). "Player skill modeling in Starcraft II". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 9. 1, pp. 2–8.

Ba, Jimmy and Rich Caruana (2014). "Do Deep Nets Really Need to be Deep?" In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf.

Botea, Adi, Bruno Bouzy, Michael Buro, Christian Bauckhage, and Dana Nau (2013). "Pathfinding in Games". In: *Artificial and Computational Intelligence in Games*. Ed. by Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Vol. 6. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 21–31. ISBN: 978-3-939897-62-0. DOI: 10.4230/DFU.Vol6.12191.21. URL: http://drops.dagstuhl.de/opus/volltexte/2013/4333.

Buro, Michael and David Churchill (2012). "Real-Time Strategy Game Competitions". In: *AI Magazine* 33.3, p. 106. DOI: 10.1609/aimag.v33i3.2419. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2419.

Critch, Lucas and David Churchill (2020). "Combining Influence Maps with Heuristic Search for Executing Sneak-Attacks in RTS Games". In: *2020 IEEE Conference on Games (CoG)*, pp. 740–743. DOI: 10.1109/CoG47356.2020.9231889.

Danielsiek, Holger, Raphael Stuer, Andreas Thom, Nicola Beume, Boris Naujoks, and Mike Preuss (2008). "Intelligent moving of groups in real-time strategy games". In: *2008 IEEE Symposium On Computational Intelligence and Games*, pp. 71–78. DOI: 10.1109/CIG.2008.5035623.

Fan, Jianqing, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang (2020). "A Theoretical Analysis of Deep Q-Learning". In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. Ed. by Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger. Vol. 120. Proceedings of Machine Learning Research. PMLR, pp. 486–489. URL: https://proceedings.mlr.press/v120/yang20a.html.

Fang, Fei, Thanh Nguyen, Rob Pickles, Wai Lam, Gopalasamy Clements, Bo An, Amandeep Singh, Milind Tambe, and Andrew Lemieux (2016). "Deploying PAWS: Field Optimization of the Protection Assistant for Wildlife Security". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.2, pp. 3966–3973. DOI: 10.1609/aaai.v30i2.19070. URL: https://ojs.aaai.org/index.php/AAAI/article/view/19070.

Filimonov, AB, NB Filimonov, and AA Barashkov (2019). "Construction of potential fields for the local navigation of mobile robots". In: *Optoelectronics, Instrumentation and Data Processing* 55, pp. 371–375.

Hagelbäck, Johan (2016). "Hybrid Pathfinding in StarCraft". In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.4, pp. 319–324. DOI: 10.1109/TCIAIG.2015.2414447.

Hagelbäck, Johan (2012). "Potential-field based navigation in StarCraft". In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 388–393. DOI: 10.1109/CIG.2012.6374181.

Hagelbäck, Johan and Stefan Johansson (2021). "The Rise of Potential Fields in Real Time Strategy Bots". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 4.1, pp. 42–47. DOI: 10.1609/aiide.v4i1.18670. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/18670.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

Huang, Haomiao, Jerry Ding, Wei Zhang, and Claire J. Tomlin (2011). "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag". In: *2011 IEEE International Conference on Robotics and Automation*, pp. 1451–1456. DOI: 10.1109/ICRA.2011.5980264.

Jiang, Q., K. Cai, and F. Xu (2023). "Obstacle-avoidance path planning based on the improved artificial potential field for a 5 degrees of freedom bending robot". In: *Mechanical Sciences* 14.1, pp. 87–97. DOI: 10.5194/ms-14-87-2023. URL: https://ms.copernicus.org/articles/14/87/2023/.

Jong, Daeseong, Ickhwan Kwon, Donghyun Goo, and DoHoon Lee (2015). "Safe Pathfinding Using Abstract Hierarchical Graph and Influence Map". In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 860–865. DOI: 10.1109/ICTAI.2015.125.

Korf, Richard E and Michael Reid (1998). "Complexity analysis of admissible heuristic search". In: *AAAI/IAAI*, pp. 305–310.

Lara-Cabrera, Raúl, Carlos Cotta, and Antonio J. Fernández-Leiva (2013). "A review of computational intelligence in RTS games". In: *2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pp. 114–121. DOI: 10.1109/FOCI.2013.6602463.

Levy, Sharon, Wenhan Xiong, Elizabeth Belding, and William Yang Wang (2020). "SafeRoute: Learning to Navigate Streets Safely in an Urban Environment". In: *ACM Trans. Intell. Syst. Technol.* 11.6. ISSN: 2157-6904. DOI: 10.1145/3402818. URL: https://doi.org/10.1145/3402818.

Li, Yuxi (2018). *Deep Reinforcement Learning: An Overview*. arXiv: 1701.07274 [cs.LG].

Macklin, Colleen and John Sharp (2016). *Games, Design and Play: A detailed approach to iterative game design*. Addison-Wesley Professional.

Makarov, Ilya, Pavel Polyakov, and Roman Karpichev (2018). "Voronoi-based Path Planning based on Visibility and Kill/Death RatioTactical Component." In: *AIST (Supplement)*, pp. 129–140.

Mark, Dave (2019). "Modular tactical influence maps". In: *Game AI Pro 360*. CRC Press, pp. 103–124.

Mendonça, Matheus R.F., Heder S. Bernardino, and Raul F. Neto (2015). "Stealthy Path Planning Using Navigation Meshes". In: *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 31–36. DOI: 10.1109/BRACIS.2015.49.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Moldenhauer, Carsten and Nathan Reed Sturtevant (2009). "Evaluating strategies for running from the cops". In: *Twenty-First International Joint Conference on Artificial Intelligence*.

Ninomiya, Kai, Mubbasir Kapadia, Alexander Shoulson, Francisco Garcia, and Norman Badler (2015). "Planning approaches to constraint-aware navigation in dynamic environments". In: *Computer Animation and Virtual Worlds* 26.2, pp. 119–139.

Panov, Aleksandr I., Konstantin S. Yakovlev, and Roman Suvorov (2018). "Grid Path Planning with Deep Reinforcement Learning: Preliminary Results". In: *Procedia Computer Science* 123. 8th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2017 (Eighth Annual Meeting of the BICA Society), held August 1-6, 2017 in Moscow, Russia, pp. 347–353. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.01.054. URL: https://www.sciencedirect.com/science/article/pii/S1877050918300553.

Puterman, Martin L (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Sailer, Frantisek, Michael Buro, and Marc Lanctot (2007). "Adversarial Planning Through Strategy Simulation". In: *2007 IEEE Symposium on Computational Intelligence and Games*, pp. 80–87. DOI: 10.1109/CIG.2007.368082.

Sartoretti, Guillaume, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset (2019). "PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning". In: *IEEE Robotics and Automation Letters* 4.3, pp. 2378–2385. DOI: 10.1109/LRA.2019.2903261.

Sauze, Colin and Mark Neal (June 2010). "A Raycast Approach to Collision Avoidance in Sailing Robots". English. In: C. Sauze and M. Neal (2010), A Raycast Approach to Collision Avoidance in Sailing Robots, in proceedings of the 3rd International Robotic Sailing Conference, Kingston, Ontario, Canada, June 6th-10th 2010.

Stanescu, Marius, Nicolas Barriga, and Michael Buro (2014). "Hierarchical Adversarial Search Applied to Real-Time Strategy Games". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 10.1, pp. 66–72. DOI: 10.1609/aiide.v10i1.12714. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/12714.

Stern, Roni, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak (2019). *Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks*. arXiv: 1906.08291 [cs.AI].

Sturtevant, Nathan and Michael Buro (2005). "Partial pathfinding using map abstraction and refinement". In: *AAAI*. Vol. 5, pp. 1392–1397.

Sturtevant, Nathan R. (2012). "Benchmarks for Grid-Based Pathfinding". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2, pp. 144–148. DOI: 10.1109/TCIAIG.2012.2197681.

Sturtevant, Nathan R., Devon Sigurdson, Bjorn Taylor, and Tim Gibson (2019). "Pathfinding and Abstraction with Dynamic Terrain Costs". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 15.1, pp. 80–86. DOI: 10.1609/aiide.v15i1.5228. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/5228.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Tastan, Bulent, Yuan Chang, and Gita Sukthankar (2012). "Learning to intercept opponents in first person shooter games". In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 100–107. DOI: 10.1109/CIG.2012.6374144.

Tremblay, Jonathan, Pedro Andrade Torres, and Clark Verbrugge (2014). "Measuring risk in stealth games." In: *FDG*. Citeseer.

Wang, Binyu, Zhe Liu, Qingbiao Li, and Amanda Prorok (2020). "Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 5.4, pp. 6932–6939. DOI: 10.1109/LRA.2020.3026638.

Yonetani, Ryo, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki (2021). "Path Planning using Neural A* Search". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 12029–12039. URL: https://proceedings.mlr.press/v139/yonetani21a.html.