School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Generating DRL Reward Functions using Adversarial, Genetic, and Imitation Techniques

Jonathan Isaac Chukwuemeka Claussnitzer

Submitted in accordance with the requirements for the degree of
Meng Computer Science with Artificial Intelligence

2023/24

COMP3931 Individual Project

- -

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (15/05/24)*
Link to online code repository	*URL*	*Sent to supervisor and assessor (15/05/24)*

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

.

(Signature of student): 

- -

Summary

This project explores an unsupervised adversarial training strategy to enhance the training of deep reinforcement learning (DRL) agents using a combination of DRL, Imitation Learning (IL), and Genetic Algorithms (GA). Agents are developed and trained within a simulation environment modelled after the Quoridor board game, leveraging Python and TensorFlow to implement the Deep Q-Network (DQN).

Training involves generating pairs of agents that simulate gameplay, with initial behaviours informed by imitation learning to establish a foundational knowledge base. Training progresses through batch sessions with dynamic opponent rotation, culminating in a Swiss-style tournament where top-performing agents propagate their strategies.

Key Research Findings:

- **Strategic Development**: The integration of GA and IL with DRL led to the dynamic evolution of reward functions, significantly enhancing strategic decision-making.
- **Performance Improvement**: Agents trained under this strategy outperformed baseline models, demonstrating the effectiveness of the combined training methodologies.
- **Adaptability and Efficiency**: The project's approach allowed agents to adapt more quickly and effectively, improving both the efficiency and depth of the learning process.
- **Evaluation Insights**: The training strategy facilitated complex incremental reward design, a crucial aspect often challenging in traditional DRL setups.

This approach addresses the inherent challenges in DRL by defining clear success metrics and designing effective incremental rewards, thereby simplifying the training process, and enhancing the learning outcomes.

- -

Acknowledgements

I extend my deepest gratitude to the University of Leeds and the School of Computing for providing an enriching academic environment and the necessary resources that significantly contributed to this research. Special thanks are due to the University for to research databases like IEEE, which were essential for my literature review and study.

I would like to express my sincere appreciation to Dr. Arash Rabbani for his guidance and expert advice throughout the duration of this project. His mentorship was crucial in shaping the research direction and execution of this project.

My thanks also go to the developers and maintainers of TensorFlow, Pytest, and PDB, GitHub. The use of these tools enabled efficient coding, debugging, and testing phases, making the technical aspects of my work both feasible and more robust.

I am grateful for the assistance provided by large language models like ChatGPT and Gemini in the initial stages of my research, which helped clarify complex concepts.

On a personal note, I am profoundly thankful to my family and friends for their endless support and encouragement. The challenges presented by my father’s illness this past year added a significant personal strain, and the support from my loved ones was instrumental in navigating through this difficult period.

Special thanks to my colleague Alex North for nudging me and stopping from procrastinating too much.

In conclusion, I acknowledge that this work would not have been possible without the collective support of the academic tools and personal encouragement I received, and I am immensely thankful for all contributions.

- -

Table of Contents

- -

Chapter 1
Introduction and Background Research

Introduction

1.1.1 Reinforcement Learning

Reinforcement Learning (RL) is a prominent technique within the field of computing that involves training agents to adapt their behaviour to achieve predefined objectives. Specifically, it is a subset of machine learning where agents, guided primarily by a system of trial and error, learn optimal behaviours through interactions within their environment.

Observation and Action in RL

In RL, agents continuously observe their environment, which can vary from highly predictable to extremely complex and dynamic. Based on these observations, agents take actions that they predict will lead to the best outcomes. Markov Decision Processes (MDPs), detailed in Appendix C.1, provide the mathematical framework for this interaction. In an MDP, agents select actions based on observed states, aiming to optimize cumulative rewards (see Figure 1). The relevance of MDPs to RL lies in their ability to formalize decision-making processes in dynamic environments.

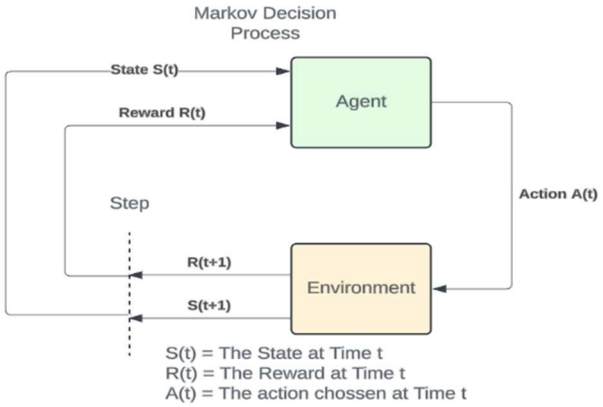


Figure 1.) Diagram Showing the Markov Decision Process (See Appendix Class Usage Diagram

1.1.2 Understanding Reward Functions

The Role of Reward Functions

Reward functions are central to the reinforcement learning process. They quantitatively describe the desirability of an outcome, providing a numerical value (the reward) for each action taken by an agent in each state. The primary goal of an RL agent is to maximize these rewards over time (Sutton & Barto, 1998), which effectively guides the learning algorithm to refine the agent's behaviour.

The Problem with Reward Functions

The problem with maximising rewards is the fact that you need to define suitable reward functions that effectively reinforce desirable actions which is not straightforward (Mnih et al., 2015). How does one instruct an agent, operating in potentially unpredictable action states, to discern 'good' behaviour? Which actions merit a reward, and should certain actions be penalized even if they might lead to more favourable outcomes later?

Jiang and Agarwal (2018) mentioned that one of the most significant challenges in designing reward functions is dealing with sparse and delayed rewards. In many real-world scenarios, rewards are not immediately available after an action is taken. This delay complicates the learning process, as the agent must determine which actions helped achieving a favourable outcome. This issue, known as the credit assignment problem, widely recognised to have been introduce by Sutton, R.S. (1984), complicates the task of designing reward functions that effectively and promptly reinforce correct behaviours.

A good reward function efficiently guides agents by clearly distinguishing beneficial actions from detrimental ones. This clarity helps the agent to learn optimal behaviours more quickly, reducing the time and computational resources required for training. Gao et all (2022) have shown that a carefully designed reward function directly leads to better performance in uncertain highway exit situations.

Furthermore, complex tasks, especially those involving multiple objectives or requiring a balance of several factors, necessitate intricately defined reward functions (Tomov et al, 2019). Good reward functions are essential for tasks such as multi-agent coordination, negotiation, and complex problem-solving, where simple reward structures may be inferior (Valk & Witteveen, 2002).

1.1.3 Project Scope

This project aims to advance the field of deep reinforcement learning (DRL) by developing an unsupervised adversarial training strategy, specifically tailored for the board game

Quoridor. The core objective is to enhance the automatic generation of reward functions, leveraging a novel combination of imitation learning (IL) and genetic algorithms (GA). This approach is expected to improve both the efficiency and effectiveness of the learning process.

By concentrating on the Quoridor game environment, the project will explore deep learning strategies in a controlled, yet complex, setting. This focus will allow for a thorough examination of the algorithms' capabilities and their potential scalability to other turn-based strategy games.

The choice to use unsupervised learning methods provides a unique opportunity to delve into the complexities and benefits of learning without explicit supervision, offering insights into more adaptive and robust artificial intelligence (AI) systems.

1.1.4 Connecting to Broader Issues

Despite, the specific scope of this project, it is expected to have a wider impact than its application to board games (Quoridor). For example, the developed strategy for reward function generation has potential applications for other strategic areas of study such as robotics and autonomous system, where finding effective rewards has shown to improve efficiency and productivity (Kutyrev, A., et al, 2022).

1.1.5 Objectives

This project aims to delineate clear, actionable objectives that will guide the subsequent research activities. The primary goals include not only advancing the theoretical framework of DRL but also applying these innovations in practical scenarios to validate their effectiveness and adaptability. This section outlines the specific objectives that will direct all research efforts towards fulfilling the overarching aim of enhancing DRL applications:

1. **Develop** an unsupervised adversarial training strategy to improve the generation and optimization of reward functions within DRL systems, using Quoridor as the testing platform.
2. **Evaluate** the effectiveness of these strategies within the simulated environment of Quoridor, assessing improvements in agent performance and learning efficiency.
3. **Analyse** the theoretical implications of these training methods, focusing on their potential to be adapted for broader AI applications in the future.

By achieving these objectives, the project will not only address the specific challenges within the realm of DRL but also contribute to the broader field of AI, potentially revolutionizing how autonomous systems are trained and operate across various industries.

1.1.6 Overview

This report provides a structured exploration of generating DRL reward functions using adversarial, genetic, and imitation techniques within the Quoridor game environment. Here is a brief guide to navigating the report:

- **Chapter 1: Introduction and Background Research:** Introduces reinforcement learning, discusses the challenges of designing effective reward functions, and reviews relevant literature.
- **Chapter 2: Methodology:** Describes the technical methodologies, development environment, and the architecture of the DRL models, including detailed discussions on the adversarial training strategy, genetic algorithms, and imitation learning.
- **Chapter 3: Results:** Presents experimental outcomes, analyses the performance of the DRL agents, and evaluates the training strategies with statistical support and visual data.
- **Chapter 4: Discussion:** Interprets the results, discusses theoretical and practical implications, addresses limitations, and suggests future research directions.
- **Appendices and References:** Provide supplementary materials and document the sources cited throughout the report for further inquiry and validation.

Literature Review

1.2.1 Benefits and Detriments of Traditional RL

Traditional RL has been successful in various applications, particularly where state and action spaces are discrete and of manageable size such as in the navigation of robotic systems in dynamic environments (Yen & Hickey, 2004). However, its application in environments with high-dimensional spaces, like rehabilitation after spinal injuries, reveals its limitations due to the expansive possible actions (Phelps et al, 2023). This complexity necessitates a robust approach like DRL that can process vast input spaces without extensive manual intervention.

1.2.2 Deep Reinforcement Learning: Challenges and Innovations

Deep Reinforcement Learning integrates the robust decision-making framework of RL with the deep learning's capability to process large and complex data structures through deep neural networks. DRL utilizes these networks to directly learn from raw input data, such as images from a video camera or complex sensor arrays, effectively mapping these high-dimensional inputs to useful action outputs. This integration allows DRL agents not only to

perform well in traditional RL tasks but also to excel in environments where the agent must learn from unstructured input data (Zhang et all, 2020).

1.2.3 Adversarial and Imitation Learning to Enhance DRL

To further refine DRL's capabilities, Imitation Learning (IL) provides baseline behaviours from expert demonstrations, setting initial parameters that guide more focused exploration (Ahlberg et al., 2023). This foundational step is complemented by adversarial methods that continuously challenge DRL agents, enhancing their adaptability and robustness by refining reward functions and policies concurrently (Fu et al., 2017).

These adversarial methods not only stabilize the training process but also ensure that DRL agents are prepared for dynamic updates and complexities in their operating environments, improving both the precision and reliability of learned strategies.

1.2.4 Genetic Algorithms for Optimization

Further optimization of DRL agents is achieved through Genetic Algorithms (GAs), which fine-tune neural network architectures and learning parameters, significantly enhancing performance and speeding convergence (Sehgal et al., 2019). This method supports the high demands of DRL by optimizing critical aspects of the learning process.

1.2.5 Reward Function Design and Optimization

Within the realm of reward function design, technologies like Reward Machines and OptionGAN play pivotal roles. Reward Machines offer a structured method to define and modify reward functions, crucial for complex environments like gaming (Icarte et al., 2020), while OptionGAN uses a GAN framework to align reward policies closely with optimal behaviours, simplifying the design process (Henderson et al., 2017).

Chapter 2
Methodology

Project Management and Development Environment

2.1.1 Programming Language and Tools

The project primarily utilises Python, a versatile and widely adopted programming language known for its ease of use and robust community support. Python serves as the backbone for both the application logic and data processing tasks.

2.1.2 Key Development Tools and Libraries

- **TensorFlow**: The project employs TensorFlow, particularly its Deep Q-Networks (DQN) module, to handle complex machine learning algorithms and model training. TensorFlow's comprehensive ecosystem allows for efficient development and scalability of the AI-driven components.
- **Pytest**: To ensure code reliability and functionality, Pytest is used extensively to automate testing. This framework simplifies the creation and execution of comprehensive test suites, helping maintain high code quality throughout the development lifecycle.
- **Git**: Version control is managed through Git, facilitating team collaboration, code integration, and change tracking. Git is essential for managing updates and ensuring that all team members have access to the latest codebase iterations. The repository for this project can be found at: https://github.com/uol-feps-soc-comp3931-2324-classroom/final-year-project-MendaciousQuark.git.
- **GitHub Copilot**: Leveraging AI (Artificial Intelligence) to assist in coding, GitHub Copilot provides real-time code suggestions and snippets based on the context of the code being written, enhancing developer productivity and code quality.

2.1.3 Environment Configuration

The Development is conducted entirely on CPUs, bypassing the need for specialized GPU hardware. This choice is primarily dictated by the project’s resource availability. The CPU-based setup, while not as fast as GPU options, provides sufficient computational power for the current requirements and ensures that the software remains accessible for development on standard hardware configurations.

2.1.4 Development Methodology

The bulk of the development process for this project was structured into two distinct sprints prior to the Easter holiday, with each sprint focusing on different key components of the final product. This phased approach was designed to manage the complexity of the project effectively and ensure a systematic progression in the development of the software.

First Sprint

The initial sprint was dedicated to creating the simulation environment for the Quoridor game. During this phase, I encountered significant challenges due to underestimating the importance of testing. This oversight led to frequent interruptions in the development process, as I needed to repeatedly address and rectify bugs that emerged during coding.

The lack of automated testing slowed down development, impacting the overall efficiency of this sprint.

Second Sprint

Recognizing the difficulties faced in the first sprint, the second sprint was primarily focused on establishing an automated testing framework. This framework was critical in supporting the maintenance of the codebase, allowing for more reliable code, and reducing the time spent on debugging. The implementation of this system marked a pivotal improvement in the project's development process, enhancing both productivity and code quality.

Adaptation Due to Personal Circumstances

Following the completion of the first two sprints, the project's structured approach was unfortunately disrupted due to personal challenges arising from my father's illness. This period required a flexible adjustment to the planned methodology. Instead of continuing with a structured sprint approach, I shifted focus to more granular, critical components of the project. This included developing specific key functions vital for the project's success, such as the tournament mechanism for the genetic algorithm, which plays a crucial role in determining the survival of agents.

2.1.5 Testing and Quality Assurance

Building on 2.1.3 Environment Configuration, this section further details the integration of Pytest into the development lifecycle of the project. The testing framework is automatically triggered by every push or pull request to the Git repository, ensuring that all new code contributions are vetted ensuring stability and functionality before integration. This proactive approach to automated testing maintains the integrity of the codebase.

2.1.6 Architecture

The software architecture of this project is fundamentally based on Object-Oriented Programming (OOP) principles, which structure the system in a modular and maintainable fashion. By encapsulating specific functionalities into classes, OOP simplifies updates, and enhances system comprehensibility.

Each class functions as a distinct module, enabling developers to manage complexities and isolate issues without affecting other parts of the system. This modularity promotes reusability, where well-defined classes can be employed across various project components or even in different projects, reducing development time and potential errors. The class usage diagram in Appendix C.2 Class Usage Diagram illustrates how these classes interact

and the relationships between them, providing a visual representation of the system's structure.

Alongside OOP, the architecture incorporates several utility functions that operate outside the confines of class-based structures. These functions are designed to perform common tasks and are accessible across various parts of the application without requiring object instantiation. By integrating these utility functions, the architecture gains flexibility and efficiency, complementing the structured approach of OOP.

This combination of OOP and standalone utility functions creates a robust and adaptable architectural framework, capable of addressing both core and peripheral requirements of the project while maintaining a streamlined codebase. The strategic application of these architectural principles ensures that the project is well-equipped to handle future enhancements and adaptations, setting a solid foundation for ongoing development and scalability.

Simulation Environment

The simulation environment for this project is Quoridor, a strategic 2-4 player board game, which has been implemented from scratch specifically for this research. Players aim to reach the opposite side of a 9x9 grid by moving in cardinal directions and strategically placing walls to obstruct opponents. Implementing Quoridor from scratch allows for complete control over the environment, which is crucial for evaluating the nuanced aspects of the developed reward functions and training strategies.

Quoridor's straightforward rules coupled with its strategic depth allow for a variety of reward structures in reinforcement learning (RL). Rewards can range from simple metrics, such as proximity to the goal, to more complex evaluations like wall placement effectiveness and control over game flow. This variety makes Quoridor an exemplary environment for testing automated reward functions and strategies developed in this project. For a description of all rewards see C.3 Reward Function Descriptions**.**

2.2.1 Environment Setup

Board Representation

The Quoridor board is represented in two formats: one for human players and one optimized for computational agents. The dual-format board representation facilitates the deployment and real-time monitoring of the adversarial training strategy. Having visual confirmation of

what the DRL agent is doing enhances the understanding of its strategic movements and decisions, allowing for better analysis and refinement of its algorithms.

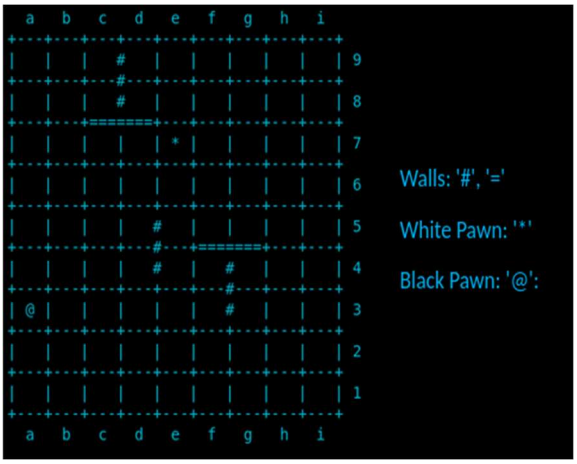
Agent Configuration

Pawns are controlled by a distinct agent. Each DRL agent has a separate DQN acting as their brain which allows multiple reward functions to be tested simultaneously. This agent setup is critical for evaluating the effectiveness of unsupervised adversarial strategies, providing insights into potential scalability and adaptability of these methods in broader AI applications.

2.2.4 Implementation

Board Implementation

Figure 2.) Visual representation of a board midgame.



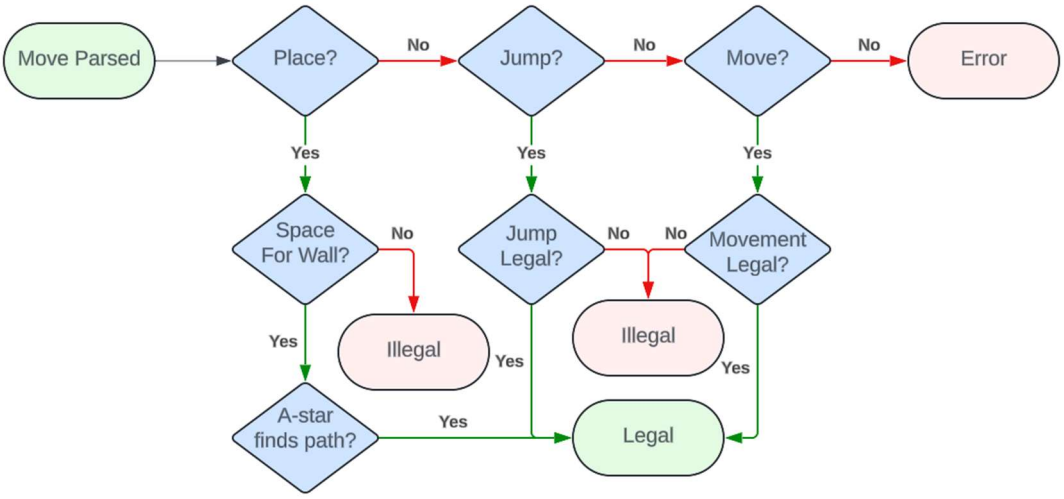
Each board consists of 81 cells, with vertical walls denoted by '#' and horizontal walls by '='. Each wall is two cells wide in accordance with Quoridor rules. The **`Cells`** in each **`Board`** store information on walls to the left and above them. This provides knowledge about all wall in the game without storing the information multiple times.

The black and white pawns are represented by '@' and '*', respectively (Figure 2). Cells are identified by a coordinate system ranging from 'A' to 'I' for columns and '1' to '9' for rows, with 'a9' referring to the top-left corner of the grid.

Move Validation and Parsing

The game supports three types of moves: standard pawn moves, jumps over an opponent, and wall placement. Each move input, whether from a user or an agent, is parsed by a Move class designed to ensure all moves conform to the game's rules. The class is structured to fail fast, rejecting impossible or illogical moves early in the validation process.

Figure 3.) High level flowchart of the move validation process



Parsed moves are stored as **`Move`** objects, which can be validated at any point using the **`validateMove`** function. As seen on Figure 3**Error! Reference source not found.**, **`validateMove`** processes each move based on its type:

1. **Standard Movement:** Moves can be made anytime, must end one cell away, and must be moving the correct colour for the given turn.
2. **Jumps:** Jumps may only be made when the opponent is in an adjacent cell, must either be diagonal or move two cells in the direction of the opponent.
3. **Wall Placements:** Walls can be placed anytime, anywhere on the board if, the player has walls remaining and placing the wall does not block either player from reaching their goal.

Wall placements are validated using the A* algorithm ensuring all moves leave a viable path to the goal for both pawns. The board is conceptualised as a graph where nodes represent positions and edges define permissible movements without walls making it simple to traverse for the A* algorithm.

Importance in AI Training

In the context of training DRL agents, the robustness of the move validation system directly impacts the quality of the agents' learning. By ensuring that only viable and logical moves are processed, the agents learn to recognize and implement strategies that are rule compliant. This setup mimics real-world conditions where decisions must adhere to predefined constraints and rules.

Environment Testing

As part of this project's automated testing framework, comprehensive unit tests were implemented for each function involved in move validation, followed by extensive integration testing. These tests are critical in verifying that the reinforcement learning (RL) models operate within accurately configured environments, ensuring that move generation and prediction mechanisms act exclusively on valid moves. To handle any anomalies during runtime, try-except blocks are employed, utilizing Python's pdb library for debugging. This setup is crucial for identifying and resolving edge cases that may not be caught during the initial testing phases.

Furthermore, the implementation of the A* pathfinding algorithm was enhanced by integrating a priority queue to improve performance. This modification was rigorously tested to validate its effectiveness. Performance tests comparing the new priority queue implementation against the original method were conducted using over 1000 random board positions. Results, gathered by using pythons `time` library, showed a significant improvement in efficiency: the priority queue implementation completed the tasks in 3.7598 seconds, whereas the previous method took 4.43136 seconds, resulting in a time improvement of approximately 15.15%. This empirical evidence underscores the success of the optimization efforts in enhancing the system's performance.

2.2.5 Challenges and Limitations

Time and expertise

Due to the limited time and the specialized expertise required, certain aspects of the project, such as enhancing the user interface for ease of human interaction, were deprioritized. The primary focus remained on developing and refining a robust simulation environment that could effectively facilitate the automated generation of new reward functions. As a result, the user experience may not be optimized, particularly in how moves are inputted into the system—currently requiring manual entry, which can be cumbersome.

Computational Resources

The computational demands of implementing an efficient pathfinding algorithm were significant, particularly because each placement of a wall requires a new execution of the A* pathfinding algorithm to validate move legality. This challenge was compounded by the need to evaluate all potential wall placements. To address these computational challenges, I introduced an optimization: tracking the last move made by each agent. By doing this, the system only recalculates legal wall placements if the last move altered the configuration of walls on the board—a natural and effective improvement that minimized unnecessary

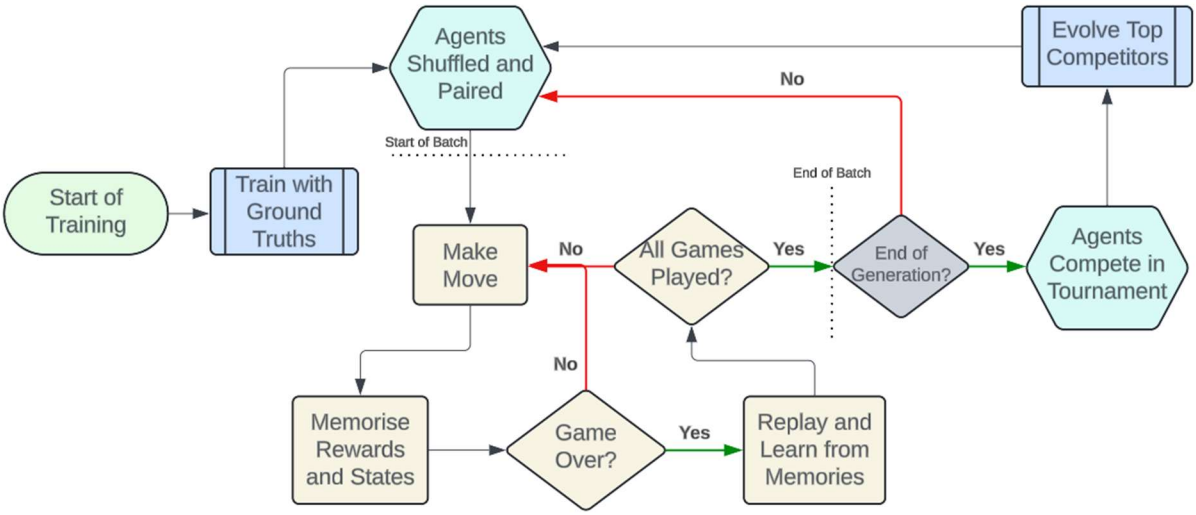
computations and enhanced performance. This approach allowed the project to adapt within the constraints of available computational resources while maintaining the integrity and speed of the simulation.

Training

2.3.1 Adversarial Training Strategy

The adversarial training strategy for this project involves a multi-phase utilising both imitation learning and genetic algorithm techniques, with specific focus on preventing overfitting to opponents and promoting genetic diversity among agents. Training consists of four main phases: the initial training; a competitive phase; and genetic propagation.

Figure 4.) High level flowchart on the training process.



Initial Training

Training commences with agents learning from ground truths to gain a basic understanding of game dynamics and decision-making processes. Ground truths, the definitive and correct outcomes, or answers within the training dataset, serve as the standard for evaluating the agents' imitation performance. This foundational training is critical as it equips the agents with necessary skills before, they face more complex game scenarios. The process of how the ground truths were obtained and more on what they are will be further discussed in 2.3.4 Imitation Learning

Following this, imitation learning, the agents will engage in gameplay sessions where the placement of walls is restricted. This modification simplifies the game environment by allowing the agents to focus primarily on movement actions. Typically, the game of Quoridor

presents a vast array of actions due to the option to place walls, which can complicate the discovery and optimization of movement strategies.

By eliminating wall placements, agents are provided with a clearer pathway to explore and master movement options, thereby enhancing their understanding of spatial navigation within the game. This approach is designed to facilitate a more focused learning experience and accelerate the development of effective movement tactics.

Competitive Phase

The competitive phase covers multiple games that together complete a batch. Each game consists of paired agents interacting and learning from each other by playing Quoridor. Before each batch, agents are shuffled and paired.

Pairing

First agents are separated into two lists (**`white_agents`** and **`black_agents`**) depending on the colour they represent: white or black. Then Python's **`random.shuffle`** method is applied on each list so that agents are randomly reordered mitigating the risk of agents learning to exploit specific opponent weaknesses.

After shuffling, in preparation for playing games, one agent from the **`white_agents`** list is paired with one agent from the **`black_agents`** list. This is done iteratively using Python's **`zip`** function, which creates tuples of corresponding elements from the two list - effectively pairing a white agent with a black agent.

Playing and Memorising

Quoridor is a turn-based game so only one agent per pair must make a move at any time. The moving agent predicts the next move. This is done by utilising its DQN (discussed in 2.3.2 The DRL Model) to process the current game state through a forward pass which transforms the state of the game into a set of Q-values. The resulting Q-values represent actions and the expected utility of taking that action.

The agent selects the action with the highest Q-value as its next move. This action selection strategy is based on the principle of maximizing the expected reward. Early in training, agents use an ε-greedy strategy to balance exploration (trying new actions) and exploitation (choosing the best-known actions). As the agent learns more about the game environment, it relies increasingly on exploitation to make decisions. Moves made and their resulting experiences (transitions from one state to another caused by an action and the reward received) are stored in a memory buffer.

This is repeated until the end of the game is reached. After which agents engage in experience replay.

Replay

Experience replay (ER) allows an agent's DQN to learn from past actions and their outcomes, which helps in stabilizing the learning process (Mnih et al., 2015). ER enables using a batch of randomly selected experiences to alter the neural network weights using backpropagation. This stops the DQN from accidently learning patterns that arise from playing against poor opponents.

Genetic Propagation

Following a predetermined batch, a generation is considered complete. The end of a generation leads to a Swiss-style tournament, which serves as both a test and a benchmark for the agents' capabilities. The outcomes of this tournament determine which agents are selected for reproduction. The tournament system is discussed in detail in section 2.3.3 Genetic Algorithm.

The limitation on the number of agents (discussed in Limitation2.3.3 Genetic Algorithm) meant that the top half of the agents from each colour were allowed to produce two children. To avoid rapidly approaching homogeneity, each selected agent produces one clone, potentially including mutations, and another offspring that retains the parent's strategic knowledge but introduces a completely new genetic set. This approach ensures that while the beneficial traits are preserved, there is also a consistent injection of new genetic material to enrich the agent pool.

By structurally integrating these phases, the training strategy attempts to prevent overfitting and promotes genetic diversity among agents, which are key factors in developing robust and adaptable AI systems. This training regime mimics real-world conditions where decisions must not only be optimal but also adhere to predefined constraints and rules.

2.3.2 The DRL Model

The architecture of the deep reinforcement learning model, specifically tailored for Deep Q-Learning, is meticulously designed to efficiently manage state inputs, and produce a corresponding vector of action values. Each value in this vector represents a potential action

the agent can take, making the model both functional and strategic.

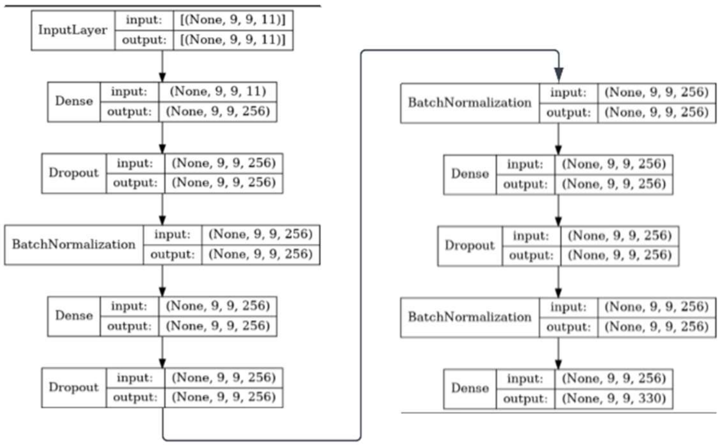


Figure 5.) Diagram showing the layers of the neural network.

At the outset, the input layer of the model plays a crucial role. It handles the state of the environment, determined by the **`state_shape`** parameter, adapting the model to different sizes of state representations based on the specific requirements of the game or application. This adaptation is essential for customizing the model to fit varied environments.

Following the input layer, the model includes three dense layers. Each layer houses 256 neurons and employs ReLU (rectified linear unit) activation functions. These functions are key in introducing non-linearity to the model, thereby enabling it to discern and learn more complex patterns within the data (Krizhevsky et al, 2012)

To combat overfitting, each of these dense layers is equipped with L2 regularization, using a lambda value of 0.01. This regularization not only helps in reducing overfitting but also ensures that the model maintains simplicity in its weight structure, which is often associated with better generalization on unseen data (Hoerl & Kennard, 2000).

Additionally, dropout is applied after each dense layer at a rate of 0.2. This technique prevents co-adaptation of neurons by randomly nullifying a proportion of output features during training iterations, which encourages the model to develop robust features that are effective in conjunction with various subsets of neurons (Hinton et al, 2012)

Batch normalization follows each dropout application. This process adjusts and scales activations, normalizing the input layer to help stabilize and speed up the neural network training process (Ioffe & Szegedy, 2015).

The model concludes with an output layer that features a dense configuration and a linear activation function. The number of neurons in this layer matches the number of actions,

outputting the Q-values for each action based on the current state input. This is critical for guiding the agent’s decision-making process.

For training, the model utilizes the Adam optimizer, renowned for its effectiveness in practice. The optimizer is set with a learning rate of 0.001 and includes a clip value of 1.0 to curb exploding gradients, ensuring stable learning progress (Kingma & Ba, 2014).

The transformation of the game state, such as the board and pawn positions in Quoridor, into a processable format for the DQN is handled by the **`BoardToStateConverter`** class. This class meticulously captures the game state, translating it into a structured three-dimensional NumPy array. This array comprehensively represents various game features, including pawn positions, wall placements, remaining walls, turn information, pathfinding metrics, and pawn adjacency. Each feature is encoded distinctly to provide a detailed map of the game state, making it possible for the neural network to make informed strategic decisions.

This detailed and structured approach to model architecture not only ensures the efficacy of the learning process but also enhances the capability of the DQN to perform optimally in complex gaming environments such as Quoridor.

2.3.3 Genetic Algorithm

The genetic algorithm (GA) emulates natural selection by creating a competitive environment where survival of the agents' genes hinges on outperforming all other members of the generation.

Limitation

The implementation of the genetic algorithm (GA) faced constraints due to hardware limitations, necessitating a small population of no more than 10 pairs of agents at any time. This restriction potentially reduced the effectiveness of the GA, as a larger population would typically allow for greater diversity in the initial set of reward functions and a higher probability of mutations during the evolutionary process.

The Process

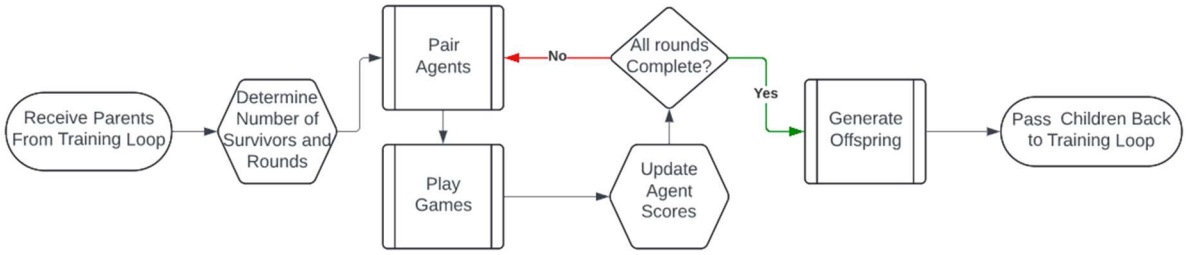


Figure 6.) High level flowchart on the evoloution process.

After initial training, which ideally involves multiple batches each focusing on distinct positions, all agents of the current generation participate in a Swiss-system tournament (Figure 6) This tournament format involves a fixed number of rounds, determined by the formula:

Rounds = ⌈log2 (Number of pairs) ⌉

Although this does not allow every agent to compete against each other, it facilitates scalability and is adaptable to larger populations.

In this application of the Swiss-system, agents are paired off in one-on-one matches following the rules of Quoridor, with each pair consisting of opposing black and white pawns. Agents are matched with opponents of similar scores to ensure fairness; for example, an agent with three wins would face another agent with a similar record.

Scoring is allocated as follows: winners receive 1 point, draws are assigned 0.5 points to each agent, and losers gain no points. While draws are typically not possible in Quoridor, they are accounted for by imposing a draw condition through a maximum number of moves, preventing indefinite game durations caused by repetitive moves by new agents. This approach is preferred as it acknowledges that drawing agents may possess equally optimal strategies, thereby avoiding the premature elimination of potentially effective reward functions.

Following the tournament, the highest-ranked agents from each colour are selected to 'survive' and pass their 'genes' onto the next generation. Offspring inherit their parents' knowledge, but the flags representing their genes have an opportunity to mutate during the replication process, introducing variability and potential enhancements in the inherited traits.

The mutation rate was consistently maintained at 5%, ensuring minimal changes from one generation to the next, thereby stabilizing the evolution of traits over time. In future research it would be valuable to alter this and determine an optimal rate of mutations.

Initially, each agent is equipped with flags representing potential 'genes'. For first-generation agents, where no flags have been previously set, the **`initialize_flags`** function is employed to randomly determine and set several flags to true. This unbiased method of generating reward functions ensures a diverse range of initial conditions for each first-generation agent.

Impact on Reward Functions

At the beginning of the training process, the population is diverse, since genes have been randomly assigned, causing a high variability in reward functions. Diversity in the population helps circumvent local maxima - situations that appear optimal but require a temporary reduction in performance to achieve superior outcomes - by enforcing at least some level of exploration over time. Especially since, due to mutations, the population does not stagnate ensuring reward functions will continue to tend towards becoming optimal for surviving the tournament.

Furthermore, as agents participate in multiple tournaments across generations, they are exposed to diverse game scenarios that challenge various facets of their strategic acumen. This pressures agents into developing adaptable strategies capable of handling the complexities of Quoridor, including varying opponent behaviours and evolving game states.

The GA fosters incremental improvements, and over successive generations, increasingly complex behaviours and strategies emerge. However, a significant drawback of this strategy surfaces due to the competitive nature of the GA: agents may learn to adapt to suboptimal strategies that, while effective against current competitors, are inferior when analysed objectively. This phenomenon highlights the critical balance required in the design of the GA to ensure that agents do not merely optimize for immediate competitive advantages but develop robust strategies that are effective in broader contexts.

2.3.4 Imitation Learning

Objective

While genetic algorithms provide a robust framework for evolving agent strategies over generations, there remains a foundational need to equip these agents with initial strategic insights that only expert knowledge can provide. This is where imitation learning comes into play, serving as a critical bridge between theoretical optimization through GA and practical strategy implementation. Imitation learning enables agents to mimic expert-level decisions, thereby accelerating the learning curve and establishing a strong baseline of strategic behaviour.

Ground Truths

Ground truths form the backbone of the imitation learning process in this project. In the context of reinforcement learning and particularly for this study, ground truths represent the optimal or most effective moves and strategies identified by an expert in various game states of Quoridor. These predetermined best actions serve as benchmarks for training the agents, providing a reliable standard against which the agents' decisions are measured.

The Expert

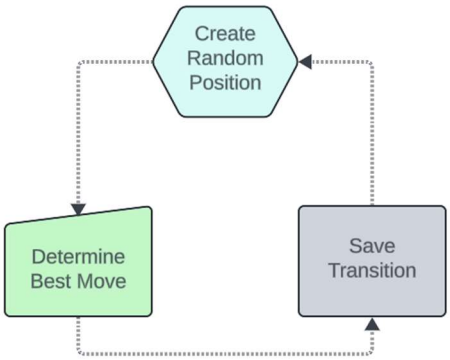
In the role of the expert, I personally took charge of data generation and labelling. While acknowledging that there are more skilled Quoridor players, the use of imitation learning in this context is not to create the perfect player but rather to provide a solid starting point for each agent's development. This initial phase acts as a foundational teaching layer that helps bridge the gap between novice and more advanced strategic engagements.

Data Generation

The process of generating ground truths involved creating and evaluating random board positions to represent a broad spectrum of potential game states (Figure 7), including which player's turn it was and how many walls remained. This randomness ensured that the training data covered a diverse array of game scenarios, preventing the models from overfitting to any specific pattern that might emerge inadvertently from the generation process.

After each board setup was ready, I, serving as the expert, would analyse the situation and determine the best possible move. This step, while crucial, was inherently challenging due to the difficulty in consistently identifying the optimal move and the unavoidable introduction of personal bias in such decisions. Repeating this procedure extensively, though not exhaustive, provided a comprehensive dataset that offered valuable insights into effective gameplay strategies.

Figure 7.) Diagram showing the iterative creation of ground truths.



4.4.5 Implementation

The integration of Behavioural Cloning was methodically executed through the **`trainWithGroundTruths`** function, designed to systematically manage, and process the training data. For a high-level overview see Figure 8.

The **`trainWithGroundTruths`** function was engineered to dynamically access a specified directory containing training files, selectively processing those that matched a predetermined naming prefix and were thus relevant for training. It initialised structures necessary for storing board states, pawn configurations, and game-specific details essential for reconstructing realistic training scenarios. A replay queue for each agent was also set up, capturing detailed records of each agent's actions and the outcomes, which were crucial for later learning phases.

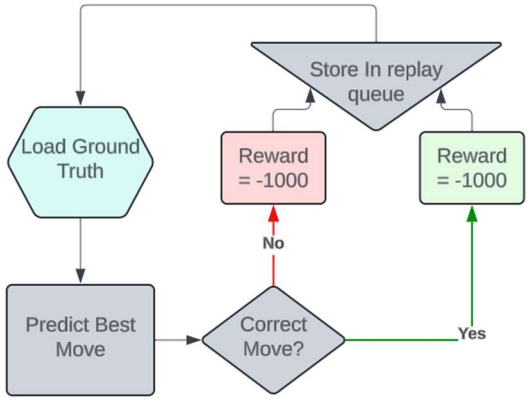


Figure 8.) Diagram showing the behavioural cloning process.

The board state was converted into a format that was digestible for the DQN, which allowed each agent to compute and execute what it determined to be the best strategic move under

those circumstances. These actions were then evaluated against the expert's decisions, with correct actions being rewarded and errors penalized, reinforcing the learning process.

The reward system was straightforward but effectively underscored the importance of accurately mimicking expert moves, thereby solidifying the agents' ability to replicate successful strategies. Each experience—comprising the state before the move, the action taken, the outcome, and the resulting new state—was stored in the agents' replay queues. This collection of experiences formed a valuable dataset that was later utilized in the reinforcement learning phase, allowing each agent to refine their strategies and improve their decision-making skills.

Chapter 3
Results

Overview of Experimental Setup

3.1.1 The Experiment

The evaluation strategy involves conducting 10 Swiss-style tournaments among different experimental agent generations, as well as baseline agents, to gauge their average performance. The tournaments include the top three agents from each participating generation. Baseline agents will consist of two entities: one playing as black and the other as white.

The simulation environment, outlined in detail in the Simulation Environment section, involves the game of Quoridor. Each round in the tournament starts with the game set in its standard position, where black and white pawns are positioned directly opposite each other at the centre of their respective rows.

Agent performance will be assessed based on the scoring system, where a win awards +1 point, a draw +0.5 points, and losses 0 points. The ranking of agents will be determined by their cumulative score and the number of rounds they have participated in. A lower number of rounds with a higher score suggests that the agent has found a strategy that can consistently beat other agents with more optimal moves.

3.1.2 Experimental Agents

The experimental agents that have been chosen to compete have four variations as shown in Table 1.

Table 1.) Agent types and their descriptions.

Name	Descriptions
Type 1	• Evolve after 4 episodes. • Each episode consists of 5 batches. • Total length of generation 20 batches.
Type 2	• Evolves after 3 episodes. • Each episode consists of 5 batch. • Total length of generation 15 batches
Type 3	• Evolves after 2 episodes. • Each episode consists of 5 batches. • Total length of generation 10 batches.
Type 4	• Evolves after 1 episode. • Each episode consists of 5 batch. • Total length of generation 5 batches.

Type 1 Agents, with their extensive evolution cycle of 20 batches across four episodes, are designed for mastering complex board positions in Quoridor, where nuanced strategy and deep analysis are essential. Their prolonged interactions before evolving should enable them to develop sophisticated strategies in positions like previously experienced situations, making them highly effective in consistently challenging or intricate game scenarios.

Type 2 Agents progress after 15 batches over three episodes, providing a balanced approach to strategy development. These agents are well-suited for Quoridor positions that require both rapid decision-making and strategic depth, enabling them to adapt effectively to a variety of game situations without sacrificing the quality of their strategic execution.

Type 3 Agents, with a swift evolution cycle completing a generation in just 10 batches, are expected to adapt rapidly to changing game dynamics. They should excel in adapting to new or unexpected board configurations in Quoridor but may lack the depth in strategy that Type 1, and Type 2 agents develop.

Type 4 Agents evolve the quickest, with only five batches per generation, making their DQN less biased towards any specific board position. This allows them to be highly adaptable and responsive to a wide range of game scenarios in Quoridor, though they may not optimize their play as effectively in any single, complex board setup.

3.1.3 Baseline Agents

The baseline agents utilize the same DQN architecture as the experimental agents depicted in Figure 5. The primary distinction between these agents lies in their training methodologies and the structure of their respective reward functions.

Training

The training process for the baseline agents is straightforward and involves only two agents, one representing black and the other white. Training commences with the initialization of each agent. Unlike the experimental agents, which undergo behavioural cloning based on ground truths (see 2.3.4 Imitation Learning), the baseline agents bypass this stage and directly engage in gameplay against each other. Importantly, there is no evolutionary process applied to the baseline agents.

The design of the baseline agents serves as a benchmark to evaluate whether the experimental agents can match or surpass the hand selected rewards of the baseline agents.

Rewards

The three rewards selected for the baseline agents are integral to their strategic learning and performance in Quoridor, each serving distinct purposes:

1. ***Opponent's Distance to the Goal****: This reward is critical as it measures the opponent's proximity to victory, using the A** algorithm to ensure the calculation considers the most efficient paths. It helps the agent assess the urgency of the game state and encourages strategic blocking and adaptive gameplay based on the opponent's position.
2. **Agent's Own Distance to the Goal Row**: Focused on the primary objective of reaching the opposite row, this reward informs the agent how close it is. By not considering walls, it challenges the agent to continuously evaluate and minimize its travel distance, fostering a balance between offensive advancement and defensive positioning.
3. **Outcome of the Game State (Win/Loss)**: Directly tying the learning process to the final game outcome, this binary reward reinforces effective strategies and penalizes unsuccessful ones. It provides a clear and strong feedback mechanism, making it an essential tool for teaching the agent the goal of winning, thereby aligning learning objectives with game-winning strategies.

These rewards collectively ensure that the agents develop a well-rounded understanding of both offensive and defensive aspects of gameplay, optimizing their strategies to align with the core objectives of Quoridor.

Results Analysis

For comprehensive data, refer to Appendix C.4 Full Tournament Table where you can find a table showing the performance of all agents. You can also find tables containing statistical analysis of the data.

3.2.1 Correlation Heatmap

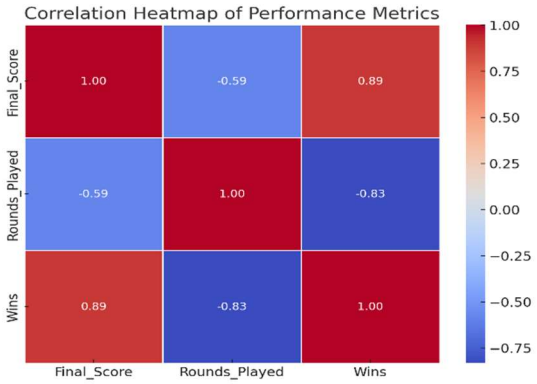


Figure 9.) Correlation heatmap of Performance Metrics

The correlation heatmap (Figure 9) provided a quantitative analysis of relationships between key performance metrics. The analysis showed that there is an obvious correlation between wins and final scores, the relationship between rounds played and final scores was less pronounced. This suggests that simply surviving longer in the game does not assure a higher final score, emphasizing the importance of effective strategy over mere endurance.

3.2.2 Heatmap Analysis

The heatmap (Figure 10) of Mean Final Scores by Agent Type and Colour revealed significant variations in performance across different agent types and colours. Specifically, Type 4 agents, particularly those coloured white, exhibited higher mean final scores compared to other types. In contrast, Baseline agents showed lower mean scores, indicating a less effective strategy under the tested conditions.

The Mean Rounds Played heatmap (Figure 10) indicated that Type 2 agents, both black and white, participated in a higher number of rounds on average, suggesting a more defensive or enduring strategy compared to other types.

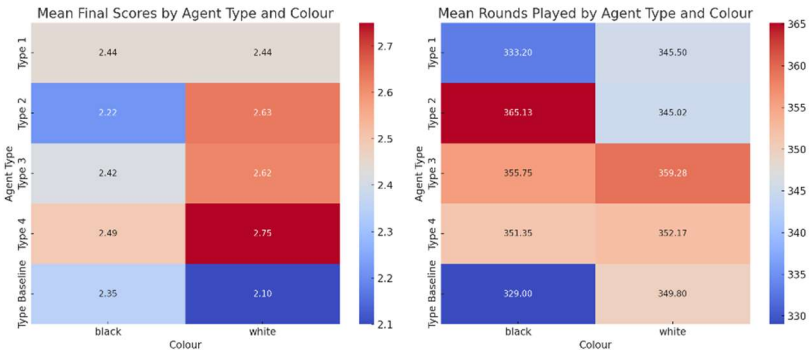


Figure 10.) Heatmaps showing how agent types performed.

3.2.3 Scatter Plot Analysis

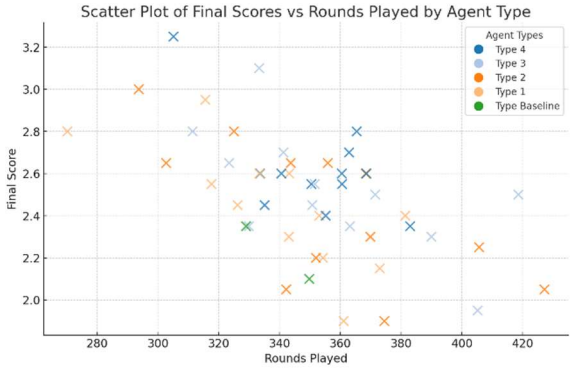


Figure 11.) Final scores vs Rounds played.

The scatter plot (Figure 11) of Final Scores versus Rounds Played highlighted a moderate correlation between these two metrics, suggesting that longer gameplay might not necessarily lead to higher scores. Each data point's colour corresponded to the agent type, displaying a broad distribution of scores and rounds, yet revealing clustering by agent type, which points to inherent performance characteristics tied to specific agent designs.

Chapter 4
Discussion

Interpretation of Results

4.1.1 Enhanced Performance through Strategic Development

The experimental agents' superior performance over the baseline agents can be attributed to the strategy developed by the project, which involved dynamic development of their reward functions and the ability to imitate expert strategies. This dual approach enabled them to

effectively prioritize early movement towards the goal—an advantage in gameplay, given the agents' observed tendency to focus initially on wall placements due to the multitude of available options for each move.

4.1.2 Validation Through Data Visualization

This strategic focus is corroborated by insights from the correlation heatmap (Figure 10) and scatter plot Figure 11), which illustrate the benefits of direct, goal-oriented gameplay. Moreover, Type 4 agents demonstrated the greatest improvement, likely because the project's design allowed them more iterations to refine their reward functions. This is further evidenced by the fact that the best versions from all 4 experimental agent types came from their highest generation involved in the tournament (Table 2).

4.1.3 Need for Optimization of Tournament Survival Conditions

However, the survival conditions in the tournament may require optimization, as evidenced by some lower-ranked agents within generations outperforming their higher-ranked peers, suggesting that the evaluation criteria might not fully capture the agents' strategic capabilities.

Theoretical and Practical Implications

4.2.1 Theoretical Implications

The findings from this project significantly contribute to the theoretical framework of Deep Reinforcement Learning (DRL), particularly in the context of unsupervised adversarial learning environments. The success of the experimental agents demonstrates the efficacy of integrating imitation learning with dynamic reward function development, supporting the hypothesis that such a combined approach can enhance the learning efficiency and strategic depth of DRL agents. This project's approach highlights the importance of adaptive learning strategies in complex game environments, where agents benefit from initial direct strategies before advancing to more complex decision-making tasks involving multiple variables, like wall placements.

The observed preference of agents for wall placement over direct movement towards the goal underscores an inherent challenge in DRL: designing reward functions that effectively balance between short-term actions and long-term objectives. The project’s strategy addresses this by allowing agents to evolve their reward structures based on their

performance, which could lead to deeper insights into the sequential decision-making processes inherent in many strategic games and real-world scenarios.

4.2.2 Practical Implications

Practically, the project's findings have several implications for the design and application of AI across various domains. Firstly, the success of these learning strategies in a controlled environment like the Quoridor game suggests potential applications in other strategic game settings, where AI agents must balance multiple tactical considerations. Additionally, the methodologies developed could be adapted for use in robotics and autonomous systems, where machines must make complex navigational decisions that balance immediate actions with strategic planning.

The project also illustrates the potential for using DRL techniques in systems where decision-making is critical, such as in autonomous driving and logistics. In these fields, AI systems could benefit from learning frameworks that emphasize the development of initial straightforward strategies before integrating more complex tasks, mirroring the phased learning approach used in this project.

Furthermore, the insights gained into reward function development could inform the broader AI community about designing systems that are not only effective in achieving their immediate goals but also adaptable and robust against a range of operational environments. This is particularly relevant for developing AI systems that operate in dynamic or unpredictable conditions, where flexibility and the ability to learn from real-time data are crucial.

Limitations and Challenges

Training and tournament design posed notable limitations in this study. Not all agents were trained for equal generations due to logistical constraints, with baseline agents completing sessions faster than experimental ones, potentially impacting performance evaluations. The Swiss-style tournament, while efficient, limited the interactions among agents, skewing results as not all capabilities were fully tested. Furthermore, DRL models optimized for this controlled setting might not perform well in more complex environments, limiting generalization. Additionally, there is a risk of the DQNs overfitting to specific scenarios if the ground truths lack diversity, which could undermine performance in unfamiliar situations. Enhanced strategies, such as parallel training processes and adopting a round-robin

tournament format, are proposed to address these issues, though they come with their own challenges, particularly in scalability.

Future Research Directions

4.4.1 Cross-Domain Applications

The adversarial and imitation learning strategies that were key components of this project have potential applications beyond the board game Quoridor. These techniques can be particularly transformative in robotics and autonomous vehicles, where dynamic decision-making is crucial. Future research could explore adapting these strategies to improve autonomous navigation and operational decision-making in robotics, testing them in environments that mimic real-world unpredictability and complexity. Similarly, these strategies could enhance real-time decision-making in autonomous vehicles, potentially improving safety and efficiency in unpredictable traffic conditions.

4.4.2 Advanced DRL Techniques

This project's integration of adversarial learning, genetic algorithms, and imitation learning offers a robust framework for enhancing the strategic capabilities of DRL agents. Future studies could expand this approach by exploring multi-agent systems where multiple DRL agents interact within more complex environments. This extension would provide insights into the scalability and adaptability of the training methods developed. Furthermore, investigating meta-learning techniques could allow DRL agents to adapt more quickly to new tasks and environments, reducing the need for extensive retraining.

4.4.3 Enhanced Adversarial Models

The adversarial training methods developed in this project can be further enhanced by focusing on continuous learning and adaptation. Future research could implement these models in environments that experience frequent changes, such as predictive maintenance for industrial machinery or real-time bidding systems in digital marketing. Additionally, improving the resistance of these models to adversarial attacks would enhance their security and reliability, making them more robust against deceptive strategies that could compromise their effectiveness.

Conclusion

This project demonstrated the integration of Deep Reinforcement Learning (DRL) with adversarial, genetic, and imitation learning to enhance AI agent performance in the Quoridor game. Key findings include:

- **Performance Improvement:** Experimental agents exceeded baseline agent performance through dynamic reward function development and the application of expert strategies refined by genetic algorithms.
- **Strategic Advantage:** Type 4 agents, with more iterative strategy refinements, consistently showed enhanced performance, underscoring the value of iterative strategy development in complex tasks.
- **Algorithm Efficiency:** The application of genetic algorithms and imitation learning accelerated the learning curve of the agents, leading to more sophisticated behaviours and demonstrating the effectiveness of these techniques in improving decision-making in AI systems.

As this project concludes, it is important to reflect on the process and the insights gained from integrating advanced machine learning techniques with strategic gameplay. The project met its objective of enhancing DRL agent performance through innovative training strategies and opened new avenues for applying these techniques in broader AI-driven applications.

This research highlights the potential of DRL combined with adversarial and imitation learning to enhance AI interactions in complex environments, pushing the capabilities of autonomous systems and robotics. The adaptability and strategic depth developed through this project pave the way for future AI applications where decision-making is critical under dynamic conditions.

The lessons from this project extend beyond academic achievements, offering practical insights for real-world scenarios. The challenges faced and addressed during this project underscore the resilience and adaptability required in research, providing a solid foundation for future explorations.

In conclusion, I appreciate the opportunity to contribute to the field of AI and am optimistic about how these strategies will be further developed to enhance AI systems across various domains. The implications of this research are significant, potentially leading to advancements in technology that are only beginning to be explored.

List of References

1. Ahlberg, W., Sestini, A., Tollmar, K. and Gisslén, L., 2023. Generating Personas for Games with Multimodal Adversarial Imitation Learning. IEEE Conference on Games (CoG), Boston, MA, USA, 2023, pp. 1-8, doi: 10.1109/CoG57401.2023.10333167.
2. Fu, J., Luo, K., & Levine, S., 2017. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. ArXiv, abs/1710.11248.
3. Gao, X., Li, X., Liu, Q., Li, Z., Yang, F., & Luan, T., 2022. Multi-Agent Decision-Making Modes in Uncertain Interactive Traffic Scenarios via Graph Convolution-Based Deep Reinforcement Learning. Sensors (Basel, Switzerland), 22. https://doi.org/10.3390/s22124586.
4. Henderson, P., Chang, W., Bacon, P., Meger, D., Pineau, J., & Precup, D., 2017. OptionGAN: Learning Joint Reward-Policy Options using Generative Adversarial Inverse Reinforcement Learning. ArXiv, abs/1709.06683. https://doi.org/10.1609/aaai.v32i1.11775.
5. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.
6. Hoerl, A. E., & Kennard, R. W. (2000). Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics, 42(1), 80–86. https://doi.org/10.2307/1271436.
7. Howard, R. (2002). Comments on the Origin and Application of Markov Decision Processes. Oper. Res., 50, pp. 100-102. https://doi.org/10.1287/opre.50.1.100.17788.
8. Icarte, R., Klassen, T., Valenzano, R., & McIlraith, S., 2020. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. J. Artif. Intell. Res., 73, pp. 173-208. https://doi.org/10.1613/jair.1.12440.
9. Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv preprint arXiv:1502.03167.
10. Jiang, N. and Agarwal, A., 2018. Open Problem: The Dependence of Sample Complexity Lower Bounds on Planning Horizon. Proceedings of the 31st Conference on Learning Theory, in Proceedings of Machine Learning Research, 75, pp. 3395-3398. Available at: https://proceedings.mlr.press/v75/jiang18a.html.
11. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.
12. Krizhevsky, A., Sutskever, I. and Hinton, G., 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: Proceedings of the 25th International Conference

on Neural Information Processing Systems - Volume 1, NIPS'12. Lake Tahoe, Nevada. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

13. Kutyrev, A., Kiktev, N., Jewiarz, M., Khort, D., Smirnov, I., Zubina, V., Hutsol, T., Tomasik, M., & Biliuk, M., 2022. Robotic Platform for Horticulture: Assessment Methodology and Increasing the Level of Autonomy. Sensors (Basel, Switzerland), 22. https://doi.org/10.3390/s22228901.
14. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., & Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature, 518, pp. 529-533. https://doi.org/10.1038/nature14236.
15. Osaki, S., & Mine, H. (1968). Linear programming algorithms for semi-Markovian decision processes. Journal of Mathematical Analysis and Applications, 22, pp. 356-381. https://doi.org/10.1016/0022-247X(68)90178-9.
16. Phelps, N., Marrocco, S., Cornell, S., Wolfe, D., & Lizotte, D., 2023. Reinforcement learning in large, structured action spaces: A simulation study of decision support for spinal cord injury rehabilitation. ArXiv, abs/2310.14976. https://doi.org/10.48550/arXiv.2310.14976.
17. Schaul, T., Quan, J., Antonoglou, I. and Silver, D. (2015) 'Prioritized experience replay'. *CoRR*, [Preprint]. Available at: http://arxiv.org/abs/1511.05952 (Accessed: 30.04.24).
18. Sehgal, A., La, H., Louis, S., and Nguyen, H., 2019. Deep Reinforcement Learning Using Genetic Algorithm for Parameter Optimization. Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 2019, pp. 596-601. https://doi.org/10.1109/IRC.2019.00121.
19. Sutton, R. S. (1984). Temporal Credit Assignment in Reinforcement Learning. Doctoral dissertation, University of Massachusetts. Available at: https://scholarworks.umass.edu/dissertations/AAI8410337 Accessed 23.04.2024.
20. Sutton, R.S. and Barto, A.G., 1998. Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press.
21. Timm, I., Bogon, T., Lattner, A., & Schumann, R. (2008). Teaching Distributed Artificial Intelligence with RoboRally. In *Proceedings of the Workshop on Education and Research in Computer Aided Architectural Design in Europe*, pp. 171-182. https://doi.org/10.1007/978-3-540-87805-6_16.

22. Tomov, M., Schulz, E., & Gershman, S., 2019. Multi-task reinforcement learning in humans. Nature Human Behaviour, 5, pp. 764 - 773. https://doi.org/10.1038/s41562-020-01035-y.
23. Valk, J., & Witteveen, C., 2002. Multi-agent Coordination in Planning. pp. 335-344. https://doi.org/10.1007/3-540-45683-X_37.
24. White, C., & White, D. (1993). Markov decision processes. https://doi.org/10.1002/0471667196.ess1539.pub2.
25. Yen, G., & Hickey, T., 2004. Reinforcement learning algorithms for robotic navigation in dynamic environments. ISA transactions, 43(2), pp. 217-30. https://doi.org/10.1016/S0019-0578(07)60032-9.
26. Zhang, D., Yin, C., Zeng, J., Yuan, X., & Zhang, P., 2020. Combining structured and unstructured data for predictive models: a deep learning approach. BMC Medical Informatics and Decision Making, 20. https://doi.org/10.1186/s12911-020-01297-6

Appendix A
Self-appraisal

A.1 Critical self-evaluation

Original Project Aim and Objectives

The initial aim of the project was to explore the application of reinforcement learning techniques in the domain of medical image segmentation. The objectives set to achieve this aim included:

- Developing a comprehensive understanding of existing medical image segmentation methods.
- Implementing and fine-tuning reinforcement learning algorithms suitable for medical image analysis.
- Evaluating the performance of the reinforcement learning models against established benchmarks.
- Investigating the impact of varying parameters on the model's segmentation accuracy.

However, only the first objective was fully met, as I gained a deeper, more comprehensive understanding of medical image segmentation methods. Midway through the year, my interests shifted, leading me to redirect my focus to a different yet compelling area of study.

Reason for Project Shift

The shift in the project's direction was driven by a growing interest in a more engaging and personally intriguing topic—developing methods for generating Deep Reinforcement Learning (DRL) reward functions. This pivot was motivated by the realization that exploring this new area could offer more substantial personal and academic fulfilment.

New Project Objectives

Following the change in focus, the project's objectives were realigned to:

- Develop an unsupervised adversarial training strategy to improve the generation and optimization of reward functions within DRL systems, using Quoridor as the testing platform.
- Evaluate the effectiveness of these strategies within the simulated environment of Quoridor, assessing improvements in agent performance and learning efficiency.
- Analyse the theoretical implications of these training methods, focusing on their potential to be adapted for broader AI applications in the future.

Achievement of New Objectives

I successfully developed an unsupervised training method and evaluated the effectiveness of these strategies in the simulated environment. The analysis of the potential broader impact of these training methods was initiated but could have been expanded further. While substantial progress was made, there remains room for deeper exploration and refinement to fully harness the theoretical and practical implications of the new strategies.

Reflections and Future Directions

The transition between project objectives provided valuable insights into adaptive project management and personal interests alignment within academic research. For future work, I plan to extend the theoretical analysis and explore additional applications of the developed methods in different AI domains. This continued effort will aim to enhance the foundational work laid out in this project and contribute further to the field of AI research.

A.2 Overview of Project Execution

Project Phases

The project was meticulously structured into several key phases:

4. **Literature Review**: An extensive review of existing literature was conducted to ensure a robust foundation of knowledge and to identify gaps in current methodologies that the project could address.
5. **Model Development**: Leveraging Python and TensorFlow, development focused on crafting and refining deep learning models tailored to the project's unique requirements.
6. **Testing and Validation**: Rigorous testing was conducted using PyTest to validate the models' effectiveness and reliability, ensuring they met the project's standards.

Development Environment

The development environment was carefully prepared to meet the high demands of training sophisticated deep learning models. This preparation was critical to ensure technical feasibility and robust execution of the project phases.

Methodology and Design Choices

- **Unsupervised Learning Techniques**: The decision to employ unsupervised learning techniques was driven by a critical analysis of the AI landscape, specifically the need to reduce dependence on labelled data. This approach aligns with current trends aiming to enhance model training efficiency and scalability.
- **Genetic Algorithms**: The integration of genetic algorithms aimed to augment the adaptability of learning models. This innovative approach pushed the boundaries of traditional reinforcement learning strategies by enabling more dynamic adjustment to the models' training processes.
- **Imitation Learning**: The implementation of imitation learning was a strategic choice designed to transfer expert knowledge directly to AI agents. This method facilitated a quicker convergence to effective strategies, reducing the computational costs typically associated with training models from scratch. Initial experiments validated this approach, demonstrating enhanced learning speed and strategic depth when agents were pre-trained with expert moves.

A.3 Personal Execution and Challenges

Throughout the project, my commitment to adapting to unexpected challenges was crucial in maintaining the progress and integrity of the research. Hardware limitations and personal circumstances tested my flexibility and resourcefulness, compelling me to adjust the project

scope and methodologies accordingly. This adaptability was vital in ensuring that the research did not merely continue but thrived under these changed conditions.

Soft Skills Development

The project significantly enhanced my research capabilities. I honed my skills in identifying credible sources, extracting pertinent information, and synthesizing knowledge across various domains to direct my project effectively. This ability to navigate complex information landscapes has become a cornerstone of my academic proficiency.

In parallel, the process of composing this report has been transformative for my formal writing skills. I have learned to organize complex information in a clear, accessible, and logically coherent manner, which is indispensable for both academic and professional communication. My advanced referencing skills have also improved, bolstering the credibility and scholarly value of my work.

Additionally, visual communication was a focal point; I developed the ability to create clear and informative diagrams that effectively convey the intricacies of my project. These diagrams have proven instrumental in illustrating complex concepts and enhancing the overall understanding of my research.

Practical Skills Enhancement

My proficiency in Python has seen remarkable growth through the project. I have mastered advanced coding techniques such as using list comprehensions for more concise code, employing built-in functions like zip and enumerate for efficient data manipulation, and implementing deep copying techniques to better manage memory and data integrity. Furthermore, I have applied complex algorithmic solutions, including the A-star (A*) algorithm for efficient pathfinding, critical for the strategic components of the game simulations.

Software testing and debugging have also been crucial areas of development. My experience with automated testing using Pytest has significantly contributed to maintaining high code quality and reliability. While exploring markers for individual tests provided limited insights, it offered valuable learning opportunities.

Utilizing external libraries such as PDB for debugging and TensorFlow for constructing Deep Q-Networks (DQNs) has expanded my technical repertoire, providing hands-on experience in building, and deploying sophisticated machine learning models.

Knowledge Acquired

The project allowed me to deepen my theoretical understanding of AI. I have gained substantial insight into:

- Reinforcement Learning (RL) and its frameworks like Markov Decision Processes (MDPs).
- Deep Reinforcement Learning (DRL), which adapts RL to manage complex, high-dimensional environments.
- Imitation Learning (IL), focusing on agents learning from observing expert behaviours.
- Genetic Algorithms (GA), which use evolutionary processes to optimize solutions.

Conclusion

Upon the completion of this project, I am overwhelmed with a sense of pride. At the outset, the scope and complexity of the project seemed daunting. However, adopting a methodical approach allowed me to manage each challenge one step at a time. This structured planning and execution led to the successful completion of the project, a testament to the power of perseverance and strategic thinking.

The journey was not without its personal challenges, particularly coping with my father's illness—a situation beyond anyone's control that introduced significant emotional and logistical hurdles. Despite these difficulties, I remained committed to my objectives, drawing on reserves of resilience I was not previously aware I possessed.

There were moments when the easier choice might have seemed to abandon the project. Yet, each time I faced the temptation to give up, I reminded myself of the potential rewards of pushing through adversity. This experience has taught me a valuable lesson about my own capabilities and the incredible outcomes possible when I commit to seeing a challenge through to the end.

The success of this project does more than just fulfil an academic requirement; it serves as a profound personal milestone. It has instilled in me a reinforced belief in my ability to overcome obstacles and achieve excellence, regardless of the complexities or hardships encountered along the way. This project has not only advanced my academic and professional skills but has also significantly contributed to my personal growth and self-understanding.

A.3 Legal, social, ethical, and professional issues

A.3.1 Legal issues

In the realm of artificial intelligence, addressing legal issues is paramount to ensuring the responsible and compliant use of technology. Broadly, legal concerns in AI revolve around intellectual property rights, data protection laws, and compliance with specific regulations governing AI applications. Navigating these legal landscapes is crucial for developers to protect their innovations and adhere to ethical standards.

Intellectual Property (IP) Rights

Intellectual property rights ensure that creators can protect their inventions and creations from unauthorized use. In AI projects, IP rights might encompass software code, algorithms, and any content generated or used during the development process.

Data Protection and Privacy

Given that AI systems often process vast amounts of data, adhering to data protection laws like GDPR in the European Union or CCPA in California is critical. These regulations dictate how personal data should be collected, processed, and stored, ensuring privacy and security for users.

Compliance with AI-specific Regulations

Various authorities may have specific regulations that directly apply to AI technologies. Compliance with these regulations ensures that AI systems are safe, reliable, and do not harm users or the public.

Legal Considerations for the Current Project

Given the scope of the current project, where all elements have been independently developed except for the utilization of external libraries like TensorFlow and the Python programming language, the direct legal concerns might seem minimal. However, understanding and addressing potential legal issues remains important:

- **Intellectual Property**: All diagrams, graphics, and textual content have been independently created for this project, mitigating risks related to copyright infringement. Using TensorFlow and Python involves compliance with their respective licenses, but these are open-source and permit free use in academic and research projects.

- **Data Protection and Privacy**: This project does not involve personal or sensitive data, reducing concerns related to privacy laws. However, maintaining awareness of these issues is critical should the project scope expand or include data handling in the future.
- **Regulatory Compliance**: While the project does not currently fall under stringent regulatory scrutiny, future applications or deployment in sensitive areas could trigger regulatory requirements. Awareness and preparation for such possibilities are advisable.

In conclusion, though the project primarily involves self-created content and uses open-source software, understanding the broader legal implications of AI development is crucial. It ensures that as the project evolves or extends to new applications, it remains compliant and responsible, respecting both legal standards and ethical norms.

A.3.2 Social issues

Artificial intelligence (AI) technologies, particularly in fields involving deep reinforcement learning (DRL), often raise significant social issues. One of the most pressing is the "black box" nature of AI systems, where the decision-making processes are not transparent or easily understandable by humans. This opacity can lead to distrust and scepticism among users and stakeholders, particularly when outcomes have significant impacts on people's lives.

Black Box Nature of AI

AI systems, especially those driven by complex algorithms and large data sets, can be difficult to interpret. The lack of transparency in how decisions are made—what data was used, which algorithms were applied, and how these decisions are derived—can be problematic in sectors where understanding the reasoning behind decisions is crucial, such as in healthcare, finance, and law enforcement.

Accountability and Trust

The opaque nature of AI systems can also impact accountability. When undesired outcomes occur, it can be challenging to discern whether they resulted from data biases, algorithmic errors, or other factors. This complicates efforts to hold the appropriate parties accountable for mistakes or ethical lapses.

Bias and Fairness

AI systems are only as unbiased as the data they are trained on. If the training data contains biases, these can be amplified by the AI, leading to unfair outcomes. Addressing these biases is essential to ensure that AI systems do not perpetuate or exacerbate social inequalities.

Social Considerations for the Current Project

The project involves developing a training strategy that automates the generation of reward functions for DRL agents, potentially deepening the black box nature of AI systems. This approach has specific social implications:

- **Increased Opacity**: By automating the creation of reward functions, the project may further obscure the rationale behind AI decisions. Users and developers might find it even more challenging to understand why an AI system made a particular decision, as the basis for reinforcement (rewards) is generated without explicit human input or oversight.
- **Implications for Trust**: The further abstraction of decision-making processes could impact the trust users and observers place in AI systems. Ensuring that these systems are still perceived as reliable and fair is crucial, especially when deployed in critical applications.
- **Mitigation Strategies**: To address these challenges, it is essential to incorporate explanation facilities or transparency features within the AI systems. Techniques such as explainable AI (XAI) can help in demystifying AI decisions by providing insights into the data and processes that drive AI behaviours. Additionally, engaging stakeholders in the development process and maintaining clear documentation of AI decision frameworks can help enhance trust and accountability.

In conclusion, while the project streamlines the training of DRL agents, making AI development more efficient, it is crucial to balance these advancements with efforts to ensure the systems remain understandable and accountable. This balance is vital to fostering trust and acceptance of AI technologies in society.

A.3.3 Ethical issues

Ethical considerations are central to the development and deployment of AI systems, encompassing a range of issues from data integrity to the potential societal impacts of technology. Here are some of the core ethical challenges typically associated with AI:

Data Integrity and Privacy

Ensuring the ethical use of data involves maintaining the privacy and integrity of the data used in training AI systems. This includes obtaining data through fair and legal means, ensuring it is used responsibly, and protecting individual privacy.

Algorithmic Fairness

AI systems must be designed to avoid perpetuating existing biases or creating new biases, which can lead to unfair treatment of individuals based on race, gender, age, or other characteristics. Ensuring algorithmic fairness involves careful design, testing, and revision of AI models to detect and mitigate bias.

Transparency and Explainability

The ability to understand and explain AI decisions is crucial for building trust and accountability. Ethical AI development strives for transparency, enabling users and stakeholders to understand how and why decisions are made.

Impact on Employment

The automation potential of AI could significantly impact job markets, necessitating ethical considerations about the displacement of workers and the role of AI in the future of work.

Ethical Considerations for the Current Project

The specific AI project described does not involve direct interaction with user data, nor does it facilitate the development of malicious AI applications. Therefore, the immediate ethical implications might seem less pronounced. However, ethical considerations still play a critical role:

- **No Direct Interaction with User Data**: Since the project does not handle or process user data, concerns about personal data privacy and integrity are minimal. However, it is essential to maintain ethical standards in data handling if the project scope changes in the future.
- **Not Promoting Malicious AI Development**: The project does not directly promote the development of malicious AI. Nonetheless, it is vital to note that the training strategy developed can potentially be applied in the training of potentially malicious programs that can be used for things like war machines.

In conclusion, while the project may have limited direct ethical risks due to its scope and nature, it is still subject to the broad ethical standards that govern responsible AI development. Maintaining an ongoing commitment to these ethical practices ensures that

projects not only advance technological capabilities but do so with a strong ethical foundation.

A.3.4 Professional issues

Professional practices in AI development encompass a range of considerations that ensure the quality, reliability, and community engagement of technology. Here are some key professional issues commonly associated with AI:

Version Control and Collaboration

Proper use of version control systems is essential for managing changes in software development, especially in collaborative environments. It allows developers to track revisions, revert to previous versions of code, and manage contributions from multiple developers effectively.

Code Quality and Maintainability

Maintaining high standards of code quality—including readability, commenting, and adherence to coding standards—is crucial. This not only facilitates easier maintenance and scalability of the software but also ensures that others can understand, use, and build upon the code.

Community Engagement and Knowledge Sharing

Engaging with the broader community through sharing research, collaborating on projects, or participating in discussions can enhance the development of AI technologies. This engagement helps disseminate knowledge, foster innovation, and build professional networks.

Professional Considerations for the Current Project

The AI project described has incorporated several best practices that address professional issues effectively:

- **Extensive Use of Version Control**: The project utilized version control extensively, enabling simultaneous development across different operating systems, such as Windows and Linux. This practice not only enhances collaboration across diverse platforms but also ensures that all changes are well-documented and reversible if necessary.
- **Use of Virtual Environments**: Employing virtual environments has helped manage dependencies and project settings without affecting other parts of the system. This

isolation is crucial for testing and maintaining project integrity across different development stages.

- **Community Engagement and Sharing**: By planning to share the project with a broader audience, there is a significant opportunity for community engagement. This openness not only allows for feedback and collaborative improvements but also contributes to the communal pool of knowledge.
- **Academic Collaboration**: Engaging with other students and academics through research has proven beneficial for mutual learning and idea exchange. This interaction enhances the academic rigor of the project and integrates diverse perspectives into the development process.
- **Code Documentation and Readability**: Extensive commenting and clear writing within the codebase improve its readability and usability. These practices are vital for future developers who may work on the project, ensuring they can easily understand and extend the work done.

In conclusion, the project has adhered to high professional standards in its execution, which not only enhances its technical quality but also its utility, scalability, and impact within the community. These professional practices are essential for sustaining long-term project success and fostering a collaborative and innovative AI development environment.

Appendix B
External Materials

This appendix provides a record of the external materials utilized in the project, distinguishing between those components developed independently and those obtained from external sources.

External Libraries and Tools Used

- **TensorFlow**: This open-source library was used extensively throughout the project for building and training deep learning models, specifically Deep Q-Networks (DQN) utilized in the reinforcement learning components. TensorFlow provided the necessary tools and functionalities to implement complex neural network architectures efficiently.

- **Python**: The primary programming language used for the project. The decision to use Python was based on its wide acceptance in the scientific and machine learning communities, as well as its comprehensive standard library and supportive ecosystem for data analysis and manipulation.
- **Python Built-in Features**: Various built-in libraries and functions of Python were employed to handle data structures, file operations, and other routine tasks. These features are part of Python's standard library, which supports various aspects of the project without the need for additional external code.

No Use of Externally Prepared Datasets

- This project did not utilize any datasets prepared by external users. All data manipulation and generation tasks were performed using algorithms designed and implemented as part of the project or were inherently handled by the TensorFlow library.

Code and Components Developed as Part of the Project

- All the code, barring the usage of TensorFlow and Python's standard features, was written independently. This includes the development of the project's specific algorithms for managing game dynamics, AI behaviour, reward function generation, and other logic related to the simulation environment of the Quoridor game.

Preliminary Materials

- No preliminary materials, drafts, or notes provided by a supervisor, or other external sources were used in the formulation or execution of this project. All conceptual and development work was carried out independently, guided by academic research and personal study.

Conclusion

This project relied primarily on the foundational tools provided by TensorFlow and Python to create a robust AI training environment. Except for these tools, all aspects of the project, including the design and implementation of specific algorithms and operational logic, were developed independently as part of the student's work for this degree. This approach ensured a deep understanding of the subject matter and integrity in the project's execution.

Appendix C

C.1 Detailed Overview of Markov Decision Processes.

Introduction

This appendix provides a detailed explanation of Markov Decision Processes (MDPs), a foundational concept in the field of reinforcement learning. While MDPs are not directly applied within the scope of this project, the principles learned from MDPs have been crucial in developing a deeper understanding of Reinforcement Learning (RL) and its extension into Deep Reinforcement Learning (DRL). This foundational knowledge has significantly influenced the theoretical and conceptual frameworks employed in this research. Howard (2002) discusses the origin and broad application of MDPs, underscoring their importance in the development of RL theories.

Theoretical Basis of MDPs

MDPs offer a mathematical framework for modelling decision-making in scenarios where outcomes are partly random and partly under the control of a decision maker. The study of MDPs provides insight into sequential decision making, crucial for understanding more complex models used in RL and DRL:

- **States (S)**: Represents all situations or configurations the environment might be in.
- **Actions (A)**: Consists of all the choices available to the decision maker at any given state.
- **Transition Probabilities (P)**: The likelihood that taking an action aa in state ss results in transitioning to a new state $s'$and receiving a reward r.
- **Rewards (R)**: A function that assigns a numerical value (reward) following each state transition, incentivizing certain actions over others.

White and White (1993) provide a comprehensive overview of the role of transition probabilities and rewards in shaping the decision-making process within MDPs.

Mathematical Framework

The formulation of MDPs typically revolves around the Bellman equation, which is essential for understanding the recursive nature of decision making in RL:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V^{\pi}(s')]$$

Here, γ denotes the discount factor, balancing the importance of immediate and future rewards, a concept that underpins many RL algorithms. Osaki and Mine (1968) discuss the mathematical underpinnings and applications of such models in optimization and policy formulation.

Policy and Strategic Decision Making

A policy π in the context of an MDP dictates the action selection in each state. Learning optimal policies, which maximize expected rewards over time, parallels the objectives in DRL where complex policies are derived to handle higher-dimensional data:

- **Optimal Policy (π^*)**: This is the goal of an MDP, analogous to finding effective strategies in DRL for maximizing performance metrics.

Influence on Project Understanding

Although MDPs are not directly utilized in this project, understanding them has been instrumental for grasping the fundamentals of RL, which is a stepping stone to the advanced techniques used in DRL. This background supports the project's approach to exploring novel strategies in DRL.

Conclusion and Relevance

The theoretical insights gained from studying MDPs have enriched my comprehension of RL's dynamics, which are extensively applied in the DRL frameworks discussed in the main text of this document.

C.2 Class Usage Diagram

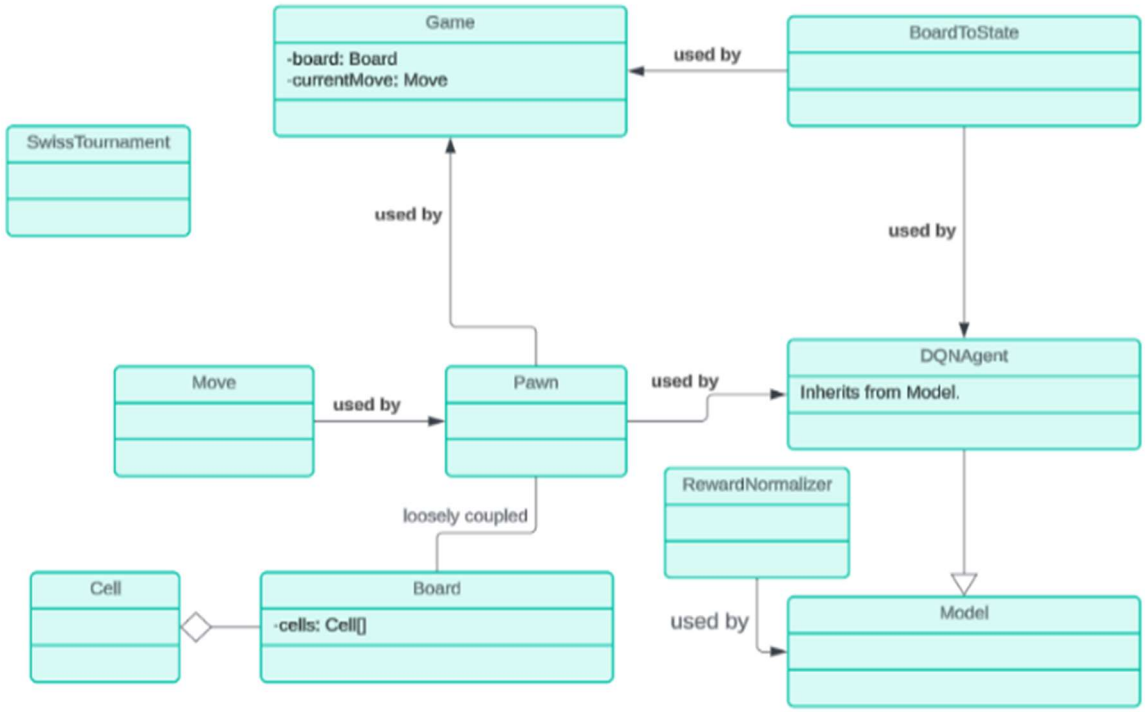


Figure 12.) Class usage diagram.

C.3 Reward Function Descriptions

This appendix section details the possible reward functions of deep reinforcement learning (DRL) agents.

Distance Difference:
Calculates the difference in the number of moves required for the agent and its opponent to win, amplified by a factor of ten to increase its impact on the agent's decision-making process. The function emphasizes speeding up the agent's progress relative to the opponents.

A-Star Distance:
Determines the number of moves required for the agent to reach its goal, not considering the opponent's position. This metric focuses solely on the agent's progress towards winning.

A-Star Distance Opponent:
Measures the number of moves required for the opponent to reach their goal, providing a metric of the opponent's progress independently of the agent's position.

Defeat or Victory:
Awards a high positive reward if the agent wins and a high negative reward if the agent loses, depending on whose turn it is. This binary reward structure sharply distinguishes between winning and losing states, reinforcing the primary objective of the game.

Distance from Goal Row:
Rewards the agent incrementally for advancing towards the goal row, with increasing rewards as the agent approaches the goal. A maximum reward is given for reaching the goal, incentivizing forward movement on the board.

Distance from Nearest Edge:
Calculates the minimum distance of the agent's position from the nearest edge of the board. This function could be used to discourage movement towards board boundaries, maintaining flexibility in positioning.

Changed Memory Reward:
Encourages diverse movement by rewarding the agent based on the number of unique positions visited over the last fifty moves. This function aims to prevent repetitive loops and encourage exploration.

Wall Difference Penalty:
Penalizes the agent if it has fewer walls remaining than the opponent, encouraging conservative use of walls and strategic planning in wall placement.

Own Wall Amount:
Returns the count of walls currently available to the agent, which could be used to factor defensive or offensive strategies into the agent's decision-making.

Opponent Wall Amount:
Provides the count of walls available to the opponent, allowing the agent to adjust its strategy based on the opponent's remaining defensive capabilities.

Enemy Wall Adjacency:
Counts how many walls placed by the opponent are adjacent to the agent, potentially useful for evaluating the effectiveness of the opponent's defensive strategies.

Self-Wall Adjacency:
Similar to Enemy Wall Adjacency but counts the walls placed by the agent that are adjacent to its own position, acting as a penalty to discourage self-limiting placements.

Distance from Nearest Wall:
Computes the distance from the agent's current position to the nearest wall on the board, which might be used to evaluate mobility or access routes critical for strategic planning.

Average Distance from Walls:
Calculates the average distance of the agent from all walls on the board, providing an overall measure of the agent's positioning relative to barriers.

Average Opponent Distance from Walls:
Measures the average distance of the opponent from all walls, offering insights into the opponent's strategic positioning relative to board constraints.

Average Distance Between Walls:
Determines the average distance between all pairs of walls, which could indicate the overall compactness or spread of barriers across the board.

Furthest Distance Between Walls:
Identifies the maximum distance between any two walls, providing a measure of the maximum open space available on the board.

Closest Distance Between Walls:
Finds the minimum distance between any two walls, useful for understanding the tightest constraints on player movement.

Distance from Opponent:
Measures the direct distance between the agent and the opponent, crucial for strategies that involve close interaction or blocking.

C.4 Full Tournament Table

The tournament involved a total of 50 agents, comprising 48 experimental agents and 2 baseline agents. The experimental agents were divided into two groups based on colour, with 24 agents assigned to each, black and white. Each agent type, detailed in Section 3.1.2 Experimental Agents, was represented by six agents from their initial generation and their highest achieved generation. All agent types were allotted identical training durations; variations in the number of generations attained reflect the inherent characteristics of each type.

Table 2.) Complete average results over 10 tournaments.

Type	Generation	Generation Rank	Final Score	Rounds Played	Wins	Overall Rank	Colour
Type 4	12	3	3.25	305.1	2.6	1	white

Type 3	7	3	3.1	333.3	2.1	2	white
Type 2	5	1	3	293.7	2.4	3	white
Type 1	5	2	2.95	315.6	2.1	4	black
Type 4	0	2	2.8	365.4	1.7	5	white
Type 2	0	3	2.8	325	1.8	5	black
Type 3	7	1	2.8	311.4	1.8	5	white
Type 1	0	2	2.8	270.2	2.2	5	black
Type 4	0	3	2.7	362.9	1.6	6	white
Type 3	0	1	2.7	341.3	1.7	6	black
Type 2	0	2	2.65	355.9	1.6	7	white
Type 2	0	3	2.65	343.7	1.9	7	white
Type 3	7	1	2.65	323.4	1.9	7	black
Type 2	5	3	2.65	302.7	2	7	white
Type 4	12	1	2.6	360.5	1.3	8	white
Type 2	5	2	2.6	368.4	1.4	8	white
Type 4	12	2	2.6	368.6	1.4	8	white
Type 1	0	3	2.6	343.2	1.5	8	white
Type 4	12	2	2.6	333.6	1.6	8	black
Type 4	12	1	2.6	340.6	1.7	8	black
Type 1	5	1	2.6	333.4	1.8	8	white
Type 4	0	1	2.55	350.5	1.3	9	white
Type 3	7	2	2.55	351.6	1.4	9	white
Type 4	0	3	2.55	360.6	1.4	9	black
Type 1	0	1	2.55	317.6	1.8	9	white
Type 3	0	2	2.5	418.6	1	10	white
Type 3	7	2	2.5	371.5	1.2	10	black
Type 3	0	1	2.45	350.8	1.4	11	white

Type 4	12	3	2.45	335.1	1.5	11	black
Type 1	5	3	2.45	326.2	1.6	11	black
Type 1	5	2	2.4	381.4	1	12	white
Type 1	5	1	2.4	353.1	1.3	12	black
Type 4	0	2	2.4	355.2	1.3	12	black
Type 3	0	3	2.35	363.2	1	13	black
Type 4	0	1	2.35	383	1.1	13	black
Type 3	0	2	2.35	329.9	1.4	13	black
Type Baseline	N/A	N/A	2.35	329	1.5	13	black
Type 3	0	3	2.3	390	1.1	14	white
Type 1	0	2	2.3	343.1	1.2	14	white
Type 2	0	2	2.3	369.9	1.2	14	black
Type 2	0	1	2.25	405.7	0.9	15	white
Type 1	5	3	2.2	354.3	0.9	16	white
Type 2	5	2	2.2	352	1.1	16	black
Type 1	0	3	2.15	373	0.9	17	black
Type Baseline	N/A	N/A	2.1	349.8	1	18	white
Type 2	5	3	2.05	427.2	0.3	19	black
Type 2	5	1	2.05	342.2	1.2	19	black
Type 3	7	3	1.95	405.2	0.6	20	black
Type 2	0	1	1.9	374.5	0.5	21	black
Type 1	0	1	1.9	361.1	1	21	black