AI for playing classical strategy games - Quoridor

Ace Shyjan

registration: 100390438

1. Introduction

1.1. Quoridor Rules

Quoridor is a two-player abstract strategy game played on a 9x9 square grid, featuring simple yet strategic gameplay where each player controls a single pawn. The primary objective is to navigate the player's respective pawn to reach the opponent's starting edge of the board before the opponent reaches theirs. On each turn, a player has two choices: move their pawn or place a fence.

1.1.1. Pawn Movement

Pawns can move one tile at a time, this can be in a forwards or backwards direction, both horizontally or vertically. Pawns are not able to be moved diagonally under any circumstances. When one pawn is positioned directly adjacent to the opponent's pawn, on their turn, they are able to "jump" over their pawn, effectively jumping two tiles in one go. This special rule will not work, if there is a fence behind the enemy pawn.

1.1.2. Fence Mechanics

Both players have access to a limited supply of ten fences. These fences once played on the board, cannot be moved around or picked back up. A fence spans over two tiles and can be placed either horizontally or vertically, however they cannot be placed in a manner to form an impossible route for either pawn. The fences can be used for many strategic plays, including to cause your opponent to backtrack.

1.2. Development

1.2.1. Godot Engine

For this project, I will be working within the **Godot Engine**. It's an open-source community driven game engine with thorough support for 2D and 3D. It provide a lightweight interface for creating applications and supports two languages: **GDScript** and **C#**. Godot has been recognised more and more recently, even being the primary focus in project by Salmela (2022). Godot has some unique features such as its **Signal System** allowing scripts to send data from one to another without wasting resources as well as a **Debugger** which allows for the user to go through the current stack frame when debugging.

1.2.2. GDScript

I will be using **GDScript** to manage general gameplay functionality within the Godot Engine. GDScript is an object-orientated language designed for Godot, offering seamless integration with the engine's core features and this integration enables efficient access to its built-in functions, and nodes. GDScript's syntax is similar to Python's as it uses indentations to organise the code and it also supports optional static typing, which can improve performance and helps reduce errors and improve productivity as it enables type hints for autocompletion

1.2.3. C#

C# will be used for the primary algorithms associated with the AI components of the game, particularly for implementing game trees, which are essential for evaluating various possible game states. C# excels in heavy computational tasks, allowing for faster evaluations of leaf nodes, which is crucial for algorithms like the Minimax Algorithm that determine optimal moves. Its static typing enables compiler optimizations, while features like strong typing and generics reduce memory overhead.

1.2.4. Other Tools

Aseprite is a lightweight graphical sprite editor. It will be used for creating and modifying any sprites and textures which are used for the game's interface. I will be using GitHub for version control and will help keep track of progress. It will also be useful as I move around between PC and Laptop. Trello will be used to organise my tasks and meetings, as well as storing any potential ideas.

1.3. Objectives

For this project, my main aim is to create a functioning game of Quoridor in 2D. The user should be able to play against a computer. Some potential features would be to let the user adjust their gameplay experience. Ideas in mind would be: adjusting the border size, computer's difficulty, number of players and adding multiplayer. Some research that will need to be done is on suitable algorithms for the computer to use.

The primary aim of this project is to develop a functional 2D implementation of Quoridor, where the user can play against a computer opponent. Additional features may include options for users to customize their gameplay experience by adjusting variables such as the **board size**, the computer's **difficulty**, the **number of players**, and possibly incorporating **multiplayer** functionality. A key aspect of this project will involve

researching and implementing suitable algorithms for the computer to effectively simulate intelligent gameplay as well as measuring their performance in game, as multiple players and a higher board size will increase the total calculation time.

2. Critical Review

This critical review will look into several AI algorithms which can be applied to Quoridor. Some algorithms I will be looking into will be: Minimax Algorithm, Monte-Carlo Tree Search (MCTS), and a Genetic Algorithm. The review will be looking into the performances of these algorithms as Quoridor is a dual-objective game - placing fences and moving the pawn.

2.1. Evaluation Functions

Evaluation functions are used to calculate a score to indicate how favourable an outcome is. In Mertens (2006), an evaluation function they used was simply the number of columns away the pawn is from the target line, meanwhile in Jose et al. (2022), the evaluation function they used was the difference in distance for the pawns to their winning side, using a Breadth-First Search, which was found to be better than a random and greedy player. It was found that in Strong (2011), it might be expected for the opponent to play a move that the evaluation function would score highly for (and low for us) and that the scoring function is unlikely to be perfect. This is because, the opponent may use moves which appear to be desirable but could later on be advantageous. This is why a higher depth/**ply** is preferred as there is more information for future game states. The issue evaluation functions is their speed and precision. As noted in Sanchez and Florez (2018), there's a trade-off between speed and accuracy in evaluation functions: a precise evaluation may be slow, while a faster one risks losing precision. Finding the right balance is essential when selecting an evaluation function that maintains both efficiency and sufficient accuracy.

2.1.1. Manhattan Distance

The Manhattan distance is the sum of the absolute differences between the coordinates of the two points. In Jose et al. (2022) it was used for their reward function for its efficiency over a Breadth-First Search (BFS), this reward function also ignored fences. In Glendenning et al. (2005), they also implemented Manhattan Distance, using two points - the pawn and the goal, and they too ignored any fences. The issue faced with this formula, is that a common strategy can be to create a route using fences which forces a pawn to move back rather than forwards.

2.1.2. Breadth-First Search

A Breadth-First Search (BFS) is a traversal algorithm using a queue data structure. It visits all nodes in a graph at the present depth before onto the nodes at the next depth-level. It was considered in Korf (1999) however they stated that the main drawback was it's memory requirements. The memory required was proportional to the number of nodes stored, which was a problem back in 1999, however nowadays the problem shouldn't be as influential as in Jose et al. (2022), they used it as one of their evaluation functions. The BFS Algorithm does have another use. In Respall et al. (2018), they used a BFS algorithm to simulate the effect of placing a fence in Quoridor. After simulating the fence placement, they ran a BFS to check that each pawn still had at least one valid path to reach its goal. Running this prevented scenarios where a pawn would be completely blocked off from the target line.

2.2. Minimax Algorithm with Alpha-Beta Pruning

A Minimax algorithm, such as covered in Plaat et al. (1996) and Strong (2011) finds the best next move for the current player. Its aim is to minimize the possible loss for a worst-case scenario whilst maximizing the player's own gain (hence *minimax*). The minimax algorithm builds a **Game Tree** which is used to represent the current game state as the root node. Each node in the game tree represents a possible game state. For each edge/connection between a parent and a child means a possible move from one game state to a future game state. In Quoridor, the child node to any parent, would be the parent's opponent's turn.

The issue with a minimax algorithm for games like Quoridor and Chess, is that due to its complex nature, it has no terminal solution meaning a game tree would be too large and complex to compute in a reasonable time frame. There are some optimisations which are done to improve the algorithms performance and efficiency. In Jose et al. (2022), they made use of a limited depth of 2 for the tree. This mean they could calculate only one set of moves for the opponent and player. A low depth of two would be that the algorithm couldn't think far ahead into the future and play like a human, but it allowed for a starting point for future work.

2.2.1. Alpha-Beta Pruning

Another optimization used was the Alpha-Beta Pruning algorithm, which improves the efficiency by eliminating branches of the game tree that are irrelevant to the final move selection. By maintaining two values—alpha, the maximum score for the maximizing player, and beta, the minimum score for the minimizing player—the algorithm can prune branches when it encounters nodes that fall outside these thresholds. This signif-

icantly reduces the number of evaluations needed, resulting in faster search times and improved AI responsiveness.

2.3. Genetic Algorithms

"Genetic Algorithms are a family of computational models inspired by evolution" - Mathew (2012). They can be used when the solution space is vast, such as for Quoridor and work by simulating the process of evolution by generating a population of possible solutions and evolving them over time to find the best/most optimal solution. The process begins by generating a population of potential solutions (**chromosomes**, which represents a different possible solution. Each chromosome undergoes evaluation through a **fitness function** which states how well it meets a criteria - this reflects the chromosomes overall effectiveness.

2.3.1. Selection, Crossover and Mutation

In this next stage, chromosomes with higher fitness scores (which are more likely to be chosen) are **selected** to be chosen for reproduction which mimics the natural selection process as the advantage traits are continued on through generations. After selection, **Crossover** occurs. This is where two parent chromosomes combine to create their offspring - this introduces a new combination of traits which allows for greater diversity among solutions. In addition, a mutation process can occur which adds random alterations to some chromosomes which allows the algorithm to explore more possibilities which crossover couldn't cover.

These iterative processes of selection, crossover, and mutation are repeated over multiple generations, gradually evolving the population toward higher fitness scores. Each generation refines the pool of candidate solutions, increasing the chances of finding a near-optimal/optimal solution.
In the paper by Jose et al. (2022), they were exploring and comparing possible algorithms for Quoridor AI. They used a minimax algorithm as a baseline and one of the algorithms they looked into was a *Genetic Algorithm*. They adopted the minimax algorithm and established a set of heuristic situation evaluation functions, one for the player and opponent: Manhattan distance, Perpendicular Pawn Distance, Breadth-First Search (BFS), Dijkstra's Shortest Path. They found that the Genetic Algorithm was an effective algorithm as it could evolve potential problems over multiple generations. However, it struggled with more strategic plays, such as wall placements due to Quoridor's complexity.

2.4. Monte-Carlo Tree Search

A study by Respall et al. (2018) focused on the application of the Monte-Carlo Tree Search (MCTS) algorithm specifically for the game of Quoridor. MCTS operates by simulating random play-outs from various possible moves, selecting those that lead to the most successful outcomes. The algorithm constructs a search tree where nodes represent different game states and edges correspond to the actions taken to transition between these states. MCTS consists of four key stages: **Selection**, **Expansion**, **Simulation**, and **Backpropagation**.

2.4.1. Selection

In the selection phase, the algorithm traverses the tree, starting from the root node and moving toward the leaf nodes. During this traversal, it balances exploration and exploitation; exploration involves examining less-frequent paths to discover potentially better outcomes, while exploitation focuses on paths known to yield favourable results.

2.4.2. Expansion

Once a leaf node is reached — indicating a point where further exploration is possible, one or more child nodes are added to the tree, provided the current state is not terminal (i.e., the game is not over). This phase increases the number of potential future game states that can be analysed, allowing for a broader range of possible outcomes to be considered in subsequent simulations.

2.4.3. Simulation

In the simulation stage, the algorithm conducts a random play-out from the newly added node(s), simulating the remainder of the game until a terminal state is reached. This involves making a series of random moves, which can help gauge the viability of the current game state. The randomness in this stage allows MCTS to explore a variety of game scenarios.

2.4.4. Backpropagration

Following the simulation, the algorithm enters the backpropagation phase, where it updates the nodes along the path taken during the traversal, all the way back up to the root node. Each node in this path has its visit count incremented, and its total reward is adjusted based on the outcome of the simulation — whether it resulted in a win, loss, or draw. This feedback mechanism helps the algorithm refine its strategy over time, as nodes representing with higher visit counts will receive more attention. This means that

MCTS does not "need any specific evaluation function" as stated in Sanchez and Florez (2018).
The paper by Respall et al. (2018) highlights that, given Quoridor's dual-objective nature—where players must both advance their pawn and strategically place walls—the MCTS approach offers a flexible way to navigate its complex decision space. However, the authors noted that further improvements in simulation strategies and overall efficiency are necessary for achieving optimal performance, suggesting that while MCTS is promising, there remains significant room for refinement in its application to Quoridor and similar games.

2.4.5. Sanchez Et Al

In a paper done by Sanchez and Florez (2018), they looked in at MCTS comparing it with a Minimax algorithm with Alpha-Beta Pruning. For their project, they chose to use MCTS as their implemented algorithm as the game tree created is asymmetrical. They stated that asymmetrical trees worked better for games like Quoridor which had a higher branching factor.

2.5. Summary

Across various reports, the minimax algorithm frequently served as a baseline for comparing AI methods in Quoridor. Among the approaches studied, MCTS demonstrated the most promising results, largely due to its adaptability in handling Quoridor's dual-objective gameplay of both advancing pawns and strategically placing walls. The flexibility of MCTS made it well-suited to exploring diverse strategies, offering an advantage over more rigid approaches. In contrast, the Genetic Algorithm struggled with Quoridor's complexity, particularly in developing strategic play and optimal wall placements, as it lacked the nuanced planning that the game demands. Despite this, implementing the Genetic Algorithm and comparing it could offer valuable insights and potentially introduce some interesting dynamics.

References

Glendenning, L. et al. (2005). Mastering quoridor. *Bachelor Thesis, Department of Computer Science, The University of New Mexico*.

Jose, C., Kulshrestha, S., Ling, C., Liu, X., and Moskowitz, B. (2022). Quoridor-ai. *ResearchGate*.

Korf, R. E. (1999). Artificial intelligence search algorithms.

Mathew, T. V. (2012). Genetic algorithm. *Report submitted at IIT Bombay*, 53.

Mertens, P. J. (2006). A quoridor-playing agent. *Bachelor Thesis, Department of Knowledge Engineering, Maastricht University*.

Plaat, A., Schaeffer, J., Pijls, W., and De Bruin, A. (1996). Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1-2):255–293.

Respall, V. M., Brown, J. A., and Aslam, H. (2018). Monte carlo tree search for quoridor. In *19th International Conference on Intelligent Games and Simulation, GAME-ON*, pages 5–9.

Salmela, T. (2022). Game development using the open-source godot game engine.

Sanchez, D. and Florez, H. (2018). Improving game modeling for the quoridor game state using graph databases. In *Proceedings of the International Conference on Information Technology & Systems (ICITS 2018)*, pages 333–342. Springer.

Strong, G. (2011). The minimax algorithm. *Trinity College Dublin*.

A. Risk Analysis

Risk	**Likelihood**	**Impact**	**Mitigation Plan**
Challenges in AI decision-making (dual objectives)	Medium	High	Focus on early AI testing with small-scale prototypes, adjust complexity gradually, and implement simpler fallback logic for AI if needed.
Difficulty in generating effective AI opponents using Genetic Algorithm	Low	Medium	Use pre-tested libraries or frameworks for Genetic Algorithm, run extensive test cases to ensure quality of opponents, and consider reducing GA scope if issues arise.
Loss of data or project files	Low	High	Use GitHub regularly
Bugs with Godot Engine	Medium	Low	Find a workaround or edit the engine as it is open source
Scope Creep	High	Low	Prioritise each feature and ensure the most important features are added first.

B. Gantt Chart for Quoridor Development

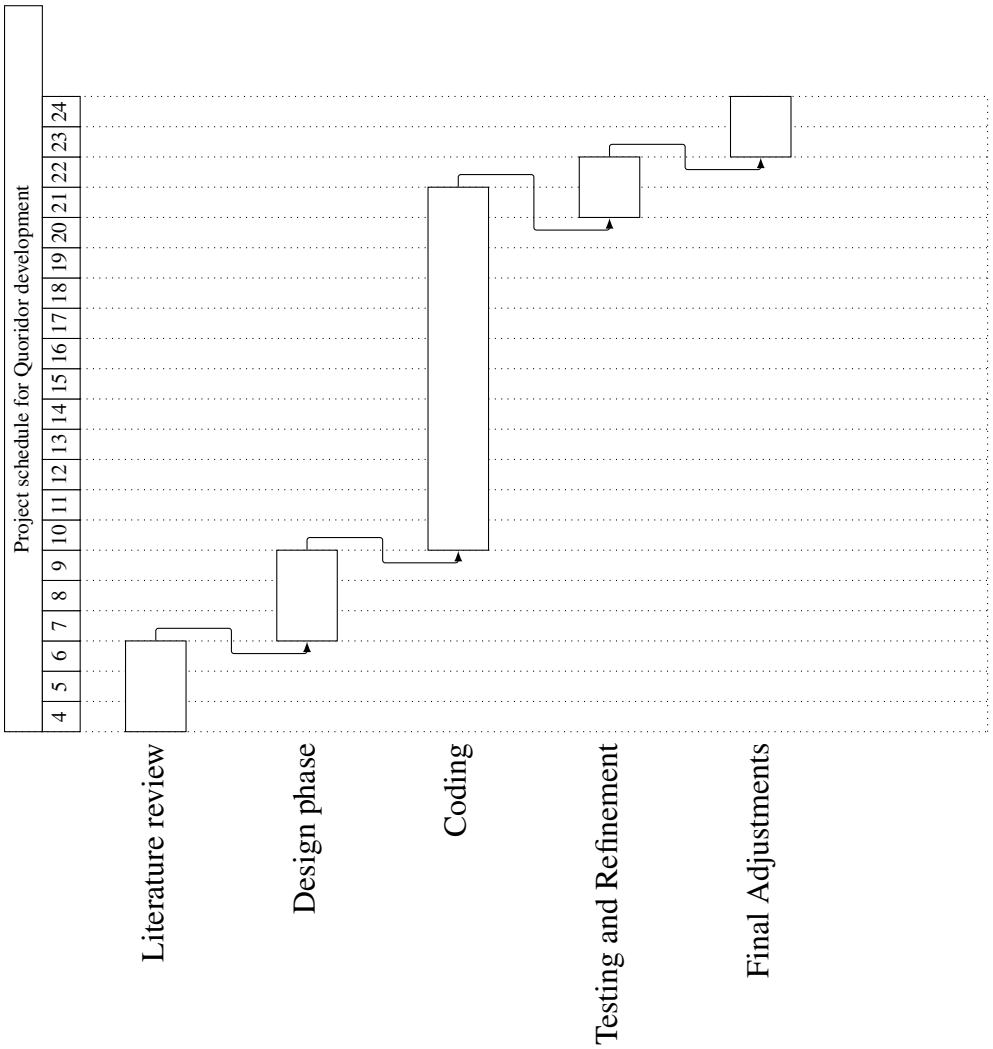

Figure 1: Quoridor Development Gantt Chart

CMP-6013Y **Student registration number: 100390438**

Literature review

Introduction: brief description of project, aims and objectives, areas of knowledge required.	First	2.1	2.2	3	Fail
Discovery of suitable quantity and quality of material.	First	2.1	2.2	3	Fail
Description of key issues and themes relevant to the project.	First	2.1	2.2	3	Fail
Evaluation, analysis and critical review.	First	2.1	2.2	3	Fail

Quality of writing

Clarity, structure and correctness of writing	First	2.1	2.2	3	Fail
References correctly presented, complete adequate (but no excessive) citations.	First	2.1	2.2	3	Fail

Planning

Risks: identification, suitable contingency planning.	First	2.1	2.2	3	Fail
Measurable objectives : appropriate, realistic, timely.	First	2.1	2.2	3	Fail
Gantt chart: legibility, clarity, feasibility of schedule.	First	2.1	2.2	3	Fail

Comments

Supervisor: Dr Michal Mackiewicz

Markers should circle the appropriate level of performance in each section.