AI for playing classical strategy games - Quoridor

Ace Shyjan

registration: 100390438

1 Introduction

1.1 Quoridor Rules

Quoridor is a classical two-player board game. It is played on a 9x9 tiled board, where each player controls a singular pawn with the end goal being to reach the opponent's starting line. On a player's turn, they are able to do one of 2 actions: move their pawn, or place a fence. Both player's have each 10 fences, which are two tiles wide. Once a fence is placed, it cannot be moved. Fences can be placed on the boundary of any tiles, but must be placed in a position which allows both player's to be able to reach their respective target location. Pawns are allowed to move one tile on their turn, this movement is limited to forwards and backwards, both horizontally and vertically. However, if two pawns are facing adjacent, the pawn in play is allowed to jump over the player, provided that there is no wall in the way, essentially allowing them to jump two tiles in one round, instead of one.

1.2 Development

For this project, I will be utilizing the Godot Engine, to manage both the game's interface and core gameplay mechanics. For the back-end development, I will be using C#. Godot does offer its own native scripting language, GDScript, and whilst the language is more tightly integrated and faster when using its regular built-in functions, C# is more suitable. C# has access to many libraries due to its popularity, compared to Godot's limited amount of plugins. In addition, C#'s performance is far superior to GDScript which is necessary for handling complex computations, such as implementing search tree algorithms. Other tools that will be likely used is Aseprite - a graphical sprite editor, which will be used for creating sprites and the user interface.

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 Minimax Algorithm with Alpha-Beta Pruning

A minimax algorithm, such as covered in Plaat et al. (1996) 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 name the player's own gain (hence *minimax*).
The algorithm builds a **Game Tree** which represents all possible moves that the player could do and the outcomes, for the current state of the game. Each node of the tree would represent a possible future game state. In the tree, the leaf nodes use an **evaluation function** to assign them a score to indicate how favourable that outcome is. 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.
Since Quoridor has no terminal solution, due to its complex nature, optimisations will have to be done. In Jose et al. (2022), they made use of a limited depth to calculate states for the near future. This meant that the algorithm wasn't able to play as well as a human, but it allowed for a starting point for future work. They used made use of an **alpha-beta pruning** algorithm. This is a form of optimisation where we prune/remove branches of the *Game Tree* which we know to be irrelevant for the final decision. This allows for the skipping of unnecessary evaluations which improves the efficiency of the search.

2.2 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. A set of candidate solutions (**chromosomes**) act as the population, where each chromosome represents a

possible solution. A **fitness function** evaluates how *fit* each solution is and assigns it a score. There is a **selection** process, which determines the population for the next stage, in here, the "fitter" chromosomes are more likely to be selected. There is a **crossover** process where two parent chromosomes create an offspring. This is a mimic of biological reproduction. Another process involved with genetic algorithms is **mutation**, where a random change in the chromosome occurs, this allows for genetic diversity allowing for the algorithm to explore further by introducing new possibilities. Theses processes are repeated across multiple generations to find a sufficient fitness score.
In 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.

2.3 Monte-Carlo Tree Search

One paper by Respall et al. (2018) dedicated the MCTS algorithm for Quoridor. MCTS works by simulating random play-outs from possible moves. It selects moves that lead to the most successful outcomes. A search tree is built where nodes represent game states and edges mean actions. MCTS cycles through four stages: **Selection**, **Expansion**, **Simulation** and **Backpropagration**.

2.3.1 Selection

In this stage, the tree is traversed, going from the root node, to the leaf nodes. When traversing, we try to balance exploration and exploitation (favouring paths that would yield better results).

2.3.2 Expansion

When a leaf node is reached, one or more child nodes are added to the tree (if it's not a terminal state). Expansion allows for more possible future game states and hence more future outcomes.

2.3.3 Simulation

Here, with the newly added node(s), we perform a random play-out/simulation of the game until a terminal state is reached. This involves making random possible moves.

2.4 Backpropagration

Then, we update the nodes long the tree path, up to the root. Each node in the path has its visit count updated and its total reward - depending on if it was a win, loss or draw. Since Quoridor is a dual-objective game, the paper by Respall et al. (2018) found that it was a promising approach as it offered flexibility in navigating its decision space, however improvements in simulation strategies and efficiency is required for optimal performance.

2.5 Summary

In all the reports, the minimax algorithm was consistently used as a baseline for comparison. MCTS, however, demonstrated the most promising results, largely due to its adaptability in handling Quoridor's dual-objective nature. On the other hand, the Genetic Algorithm struggled with this duality, as it was unable to develop strategic play. Nonetheless, implementing and comparing it could offer valuable insights and potentially introduce some interesting dynamics.

References

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

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

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.