AI for playing classical strategy games - Quoridor

Ace Shyjan

registration: 100390438

1 Introduction

1.1 Quoridor Rules

Quoridor is a 2-player board game on a 9x9 tile board. On a player's turn they are able to do one of 2 actions: place a fence, or move their pawn. Both players normally receive 10 fences each. The player is able to place a fence anywhere as long as it is possible for both pawns to reach their respective target location. The player is allowed to move both forwards and backwards, vertically and horizontally. If both players are adjacent, the current player can jump over the player, provided that there is no wall in the wall, essentially allowing them to jump 2 tiles in one round, instead of 2

1.2 Development

For this project, I will utilize the Godot Engine to manage both the game interface and core gameplay mechanics. Godot provides support for its native language, GDScript, as well as C#. I have chosen C# as the primary language for development due to its superior performance, particularly in terms of execution speed, making it more suitable for handling complex computations which will be needed for the search tree algorithms.

1.3 Objectives

The main aim for this project is to create a working game of Quoridor in which the user can play against the computer and potentially implement multiplayer into the game. Some algorithms to look into and research would be: Minimax Algorithms with Alpha-Beta Pruning, Monte-Carlo Tree Search (MCTS), Genetic Algorithms

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, MCTS, Genetic Algorithms. 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 finds the best next move for the current player, assuming that both players play optimally, using an evaluation function. 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. Alpha-beta pruning is an algorithm which can be used with Minimax to reduce the number of nodes in the search tree, which in turn improves the efficiency of the search.

2.2 Genetic Algorithms

A report was done by Jose et al. (2022) exploring possible algorithms for Quoridor and they investigated the performance and efficiency of multiple algorithms using a minimax algorithm as a baseline. One of the algorithms used were Genetic. In this report, they found that Quoridor was too complex to compute a terminal solution, so instead 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 adopted the minimax algorithm and specified evaluate functions for a max and min situation. They 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. Then, they used the genetic algorithm as the learning algorithm to optimise the search and simulate biological evolution, including replication, crossover and mutation. The weights of evaluation functions (features) were encoded as chromosomes and the algorithm learned the weight of each feature after selection, crossover and mutation. The best performed chromosomes are selected to crossover and mutate to generate the next population.
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.
Within Quoridor, MCTS has a unique challenge being that there are 2 objectives - movement and wall placement. The paper does discuss strategies on selection one of the objectives and how to simulate random games effectively. The authors implemented MCTS and compared its performance to other approaches and 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.

References

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

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.