Quoridor

Narek Khachikyan, Kima Badalyan, Ruben Galoyan

American University of Armenia

CS246: Artificial Intelligence

Professor- Monika Stepanyan

Contents

Chapter 1

Introduction

Quoridor was invented by Mirko Marchesi and first released in 1997. Conceived and developed in Italy, it immediately gained popularity due to its unique blend of strategy and simplicity. Quoridor is published by Gigamic, a French board game company known for producing high-quality and engaging games. The core of the game is to navigate through the board while placing walls to hinder opponents, making it a challenge of both pathfinding and strategic thinking. It has since become a popular game worldwide, appealing to both casual and serious board game enthusiasts.

Quoridor is a strategic board game for two players to four, with each forming a route from one end of a 9x9 grid to the other. What makes Quoridor interesting to analyze is its dual-objective nature: The pawn movements as well as strategic wall placement must be managed(Browne, 2012). There are walls, a tactical element that allows players to block opponents from their travel routes, complicating just this standard pathfinding problem(Coulom, 2006). Players often place a finite number of walls between each other, usually ten, and must move a defensive wall to block any opponents while protecting their walls (an offense).

As AI is required to include pathfinding, opponent moves anticipation, and obstacle creation in its play rule mechanism, Quoridor's complexity is a good test case for AI(Carmel & Markovitch). In particular, this project exploits these same AI techniques typically used in board games like Minimax with alpha-beta pruning and Monte Carlo Tree Search (MCTS) to solve problems of movement and obstruction, investigating how AI can overcome dynamic, opponent-based scenarios(Browne, 2012).

We developed an AI agent that makes the optimal decisions in the game of Quoridor, i.e. (choosing the best moves for each step). Advanced decision-making algorithms will be used by the AI in order to increase its chances of getting to the goal first, by strategically placing walls to slow down the opponent's progress while minimizing the AI's path(Coulom, 2006). The report is divided into sections on literature review, methodology, progress, evaluation, and conclusions. It is each section about the design, development, and analysis of the AI as it progresses towards a competitive Quoridor player.

Rules of the game:

- 2-4 players, that have the same goal - reaching the opponent's territory first.
- Each player starts at the center of his baseline. A draw will determine who starts.
- Every player has 10 walls, that cover 2 sets of 2 squares, horizontally or vertically, and cannot intersect an existing fence. Also, a fence cannot be placed that completely blocks a player's access to his/her goal.
- If the two pawns are adjacent to each other with no fence between them, the player whose turn it is can jump his/her pawn over the opponent's pawn. This jump must be in a straight line. In case after the user there is a bot blocking the jump, the movement cannot be done

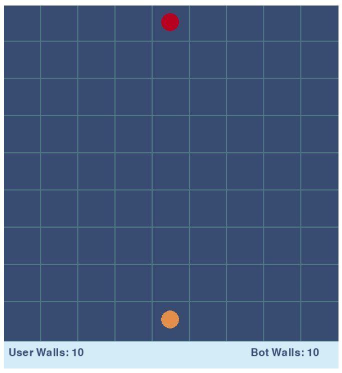

Figure 1.1: Initial Position

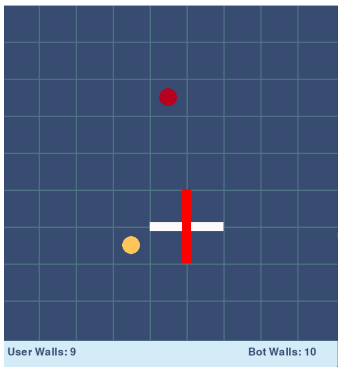

Figure 1.2: Walls can't overlap

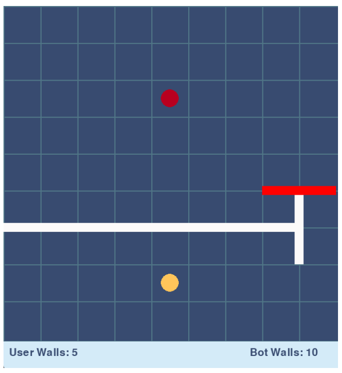

Figure 1.3: At least 1 path should be open to opponent's territory

Chapter 2

Literature Review

The AI research in strategic games has typically been considered deterministic, like Chess, or probabilistic, like Poker(Russell & Norvig, 2021). With deterministic games, AI can calculate a game tree of possible moves and calculate each path, and determine the best possible move. For example, the Minimax algorithm is a very well-known classic approach that consists simply of the fact that if the AI is to perform actions is recommended to compute all the possible actions and pick the one that will lead to minimum loss and maximum gain(Carmel & Markovitch, 1996). Alpha Beta pruning is used to eliminate branches of the game tree, hence the necessity to explore, which brings down computational effort(Browne, 2012).

Besides Minimax, there is also one well-known algorithm named Monte Carlo Tree Search. The MCTS algorithm is an alternative to Minimax, and it does well in games that have a larger search space(Coulom, 2006). MCTS explores the best outcome by playing many random plays and estimating such outcomes(Browne, 2012). For wall placement strategies in Quoridor, where we have a lot of possible placements and it's computationally expensive to explore all of them, MCTS could be useful. MCTS allows the AI to simulate the outcomes of having different wall placements and learn what wall configurations are most effective at ruining the opponent's path while still providing its own(Russell & Norvig, 2021).

The basic mechanics of the game are block and advance in Quoridor and this is a topic in some AI path-finding research. The most notable work in the area addresses pathfinding separately from wall position, but there has been little work that addresses exactly the combination of wall placement and pathfinding that Quoridor requires. Studies in practical grid-based obstacle navigation, as applied in AI for robotics problems, are shown to handle dynamic path interruptions but do so in a domain with no adversarial aspect (such as in Quoridor). This project will use these principles to bridge this gap and take advantage of strategies from both adversarial search and path optimization while working in a competitive setting(Carmel & Markovitch, 1996).

While Minimax with alpha-beta pruning is highly effective in deterministic games with well-defined outcomes, MCTS provides flexibility when dealing with larger state spaces or unpredictable outcomes. Apart from that, the branching factor is considerably less compared to the Minimax algorithm, making the algorithm cost-efficient. Unlike, MCTS the Minimax algorithm with alpha-beta training changes its branching factor dynamically. It means, that after some steps, the model will prune some paths, thus decreasing branches. The algorithm MCTS is particularly advantageous in games like Quoridor, where the outcome of each wall placement is difficult to predict due to the range of possible opponent actions. A comparative study of these two algorithms reveals that while Minimax provides clear, optimal strategies in smaller, deterministic environments, while MCTS excels in adaptability and decision-making speed. Even though MCTS is "considered" more preferable, the algorithm has some serious drawbacks. You can use the MCTS approach in two ways. First, you can generate the moves randomly starting from the first step. So, from the start of the game, you have 3 places to go(left, right, and up). The algorithm will randomly choose one of the moves which may not be the optimal one and eventually, it may lead to undesired results, i.e.(losing the game, or being stuck in an infinite loop). The second way of using MCTS is when you use some algorithm before applying MCTS. For instance, you can use the Minimax algorithm until let's say 5th step, then apply the MCTS algorithm. This approach is more reliable since the agent knows exactly where to go until some predefined step. After, it can generate random moves, which again doesn't guarantee you reach the opponent's territory first. It is worth noticing that, the MCTS algorithm is highly dependent on probabilities, which may not be justified. To handle this issue, some trigger functions should be used. However, this approach can become highly costly, which is not optimal. So, taking into consideration these scenarios, it becomes very challenging to say which algorithm performs better. For Quoridor, an ideal strategy might blend both aspects, using Minimax for short-term movement decisions and MCTS or a similar probabilistic model for wall placements.

However, our final model will be determined, when we start working practically.

In recent years, the AlphaZero algorithm shaped deterministic games by its effectiveness. After thorough research of the previous two algorithms, we came across AlphaZero and after investigating it, we can finally conclude that this is the best-performing. First, the AlphaZero uses MCTS and integrates it with reinforcement learning. This makes the algorithm, both unpredictable and very hard to beat. Then, the algorithm trains itself on its previous result, so eventually, this algorithm will be unbeaten after enough training. Later, we discussed reward functions that are related to this algorithm.

A key area in which Quoridor differs from simpler pathfinding challenges is its need for adversarial pathfinding(Carmel & Markovitch, 1996). Now in this context, AI must not only find the shortest path but also figure out how each wall placement affects what is open to its opponent. In Quoridor these concepts are very important since effective AI has to predict how to manipulate or discourage an opponent's progress(Coulom, 2006). Adversarial pathfinding is something we want to incorporate into this project by using a model to evaluate how likely an opponent's next move is and use this to decide where the walls will be.

Producing models to counter different opponent strategies is one key part of the AI's development(Carmel & Markovitch, 1996). These are 'personalities' that the AI switches according to the AI's observation of the behavior of its opponent. If, for example, the AI encounters an opponent that blocks paths with aggression, it may choose a defensive one, saving walls for itself regardless and favoring maneuverability over-aggressive strategy. However, against a passive opponent, the AI knew the AI would respond using a more aggressive blocking strategy. In simulation, these counter-strategy models are being refined to try and be as AI adaptable as possible to various play styles.

Chapter 3

Method

Wall placement strategies in Quoridor are very different depending on opponent behavior and the board layout in play. To do this, the AI has taken an approach of adaptive heuristics – to ensure that they don't prioritize certain areas, they often pick out areas on the board that intersect with opponent paths(Carmel & Markovitch, 1996). That is, calculating 'pressure points' on the grid, areas where opponent movement is likely to be constrained by walls. These points are AI-updated based on the patterns of opponent movement and focused wall placement in high-pressure areas(Russell & Norvig, 2021). Dealing with bottlenecks and intersections, the AI can guide the opponent to take longer lines or to make the same moves constantly to stop them from proceeding and to also stay on its route.

The AI combines a simulation model to predict opponent moves to boost strategic adaptability. Whenever a wall is put down, AI tries all the possible responses from the opponent — calculating an alternative route through the board and changing its strategy to be unpredictable. (Coulom, 2006). Minimax allows the AI to evaluate potential moves several steps ahead, then the algorithm adjusts based on opponent positions and possible responses. This method effectively creates a look-ahead mechanism, allowing the AI to simulate different paths and select the one that minimizes its distance to the goal row while accounting for potential obstacles(Browne, 2012). Additionally, heuristic evaluation function(s) (such as A*) will be introduced to assign scores to game states based on the AI's pawn proximity to the goal and the effectiveness of walls in blocking opponents. Each state evaluation includes weights for movement and wall placement, which can be adjusted based on the game phase or the opponent's apparent strategy.

Game State Representation: We implement the 9x9 grid as a 2D matrix. As the game evolves the matrix changes in the fly, keeping the AI always re-evaluating the paths and finding the best route.

Movement Strategy: The AI uses Minimax to evaluate potential moves—only the shortest path is considered, but in the presence of opponent-placed walls other paths may be advantageous. Alpha-beta pruning effectively cuts down on computational overhead and is great for AI because it is smart enough to know which paths to follow it ignores the branches that will yield little progress. The matrix is dynamically updated as the game progresses, allowing the AI to continuously re-evaluate paths and identify optimal routes.

Wall Placement Strategy: A second function manages wall placement based on analysis that among other things looks for the activity of the opponent and a likely path to take. We score each potential wall placement by how well it can block the opponent and allow the AI's path to stay open. For example, this could score walls by assigning higher values for walls that force the opponent to longer routes while leaving the AI's route free. Furthermore, walls are used in areas in which another opponent's path is more restricted, which 'forces' an opponent into disadvantageous positions.

Implementation Plan

- **Stage 1:** With the help of the PyGame library, we created a game environment with movement, wall placement, and victory condition functions.

- **Stage 2:** The algorithm decides the best action for the bot: movement or wall placement. It is done based on the stage of the game. The game is separated into two stages based on the number of actions done: Early and Mid-late. In the Early stage, the bot prioritizes movement over wall placement, while in the Mid-late stage, the weights are equal. Additionally, the bot moves to its goal if it is one step away from it, no matter the other factors. Furthermore, if the user is two or fewer steps away from their goal, the bot prioritizes wall placement over movement in order to block the user's path to the goal.

- **Stage 3:** We implemented a wall placement strategy, in order to determine which wall placement will have the least impact on the AI and the opponent.

How to play the game?

- If you want to move the player upwards use $\uparrow$, downwards $\downarrow$, left $\leftarrow$, and right $\rightarrow$, from the down part of your keyboard.
- If you want to place a wall, press (w).
- If you press “w”, the horizontal wall will appear in the down part on the board.
- You can move the wall, using the above-mentioned symbols.
- Using the “space” button will result in rotating the wall from horizontal to vertical.
- Press (Enter) to confirm the wall’s placement.
- If you wanted to place the wall at first and press “w”, but then you regret it, you just need to press the “m” button and make your move.

Chapter 4

Evaluation

To assess the strategic depth, efficiency, and overall performance of the Quoridor playing AI, a series of test matches were performed. We analyzed the AI's gameplay with metrics derived from 50 matches versus human opponents with different strategies. Methodology, key performance indicators, and observations are presented in this section. For decision making the AI used the Minimax algorithm, a search depth of three, and the alpha-beta pruning. The AI's win rate, move efficiency, and strategic use of walls were recorded as metrics to compare.

A statistical summary of the AI's gameplay metrics is provided in the table below:

Metric	**Value**
Total Games Played	50
Games Won	33 (66%)
Average Moves per Game	24
Minimum Moves to Win	19
Maximum Moves to Win	33
Average Walls Used per Game	8

Observation and Analysis:

1. **Wall Placement:** On average, the AI would block the opponent's path by averaging 8 walls per game. The strategy it took often forced opponents to take longer routes, and sometimes it created bottlenecks to the point that it took a long time for everybody else to get in.

2. **Move Efficiency:** For the simplest games, the AI needed about 24 moves to win on average, while for the most complex games, that number varied from 30 to 33 moves. Games higher in the standard deviation offered more than 30 moves.

3. **Win Rate:** The AI wins 66% of the time, with similar consistency. Focusing on the remaining losses, it was revealed that search depth bounds did not extend deep enough to accommodate multi-step planning scenarios.

4. **Strategic Behavior:** A heuristic function was used to guide its moves by repeatedly minimizing its distance to the goal whilst disrupting the opponent's path. This resulted in sometimes predictable behavior, however human players can exploit that advantage during a game.

What can be inferred:

- The heuristic-driven strategy of the AI for the three-depth Minimax with 3 depth results in a consistent win rate and efficient move.
- The moves-to-win metric shows variability in its value across different gameplay scenarios, indicating that the AI can adjust to different play styles of gameplay; however, its advanced foresight is limited against opponents that play strategically.
- Analysis of wall usage patterns indicates that the AI demonstrates a significant focus on disruption, often over-prioritizing defensive strategies. This tendency appears to slow down the AI's progression toward achieving its primary goal.

Chapter 5

Conclusion

This paper describes an AI agent for the Quoridor board game by focusing on strategy and making decisions. The introduction views Quoridor as one of the most challenging pathfinding and blocking games to date, which is why applying AI techniques to it has been quite intriguing. The review of related literature has shown approaches toward similar games in the past and how methods like Minimax with alpha-beta pruning are quite useful but can be enhanced for Quoridor. The methods section describes how the AI was built using the Minimax algorithm and a custom heuristic. This heuristic will help the AI to balance two important tasks: effectively moving toward the goal and blocking the opponent. These together make the AI smart and swift in making decisions. The test of the model in various conditions was a very good method of showing how well this AI will perform. Results showed that the AI plays well by winning games and using efficient moves. However, it struggles with more complex multi-step strategies in some cases.

Overall, this paper has shown that classical algorithms, when combined with well-designed heuristics, are able to provide a strong Quoridor AI. It offers a good solution for playing this game and leaves room for further improvements and experiments.

The next steps could also include neural network-based opponent modeling, meaning that the AI will learn predictive models of opponents' strategies. With this, the AI would have the ability to "learn" from every game it plays creating a feedback loop that will help refine its decision-making in every possible way. Also, other modifications, such as real-time performance modifications like real-time wall prioritization in the high-impact area may further improve the AI's competitive weighting. Difficulty scaling where the AI is pitted against progressively more sophisticated opponents is another way of testing the adaptability of the AI. It is no news that integrated AI performs effectively against basic opponents but requires further refinement to handle adaptive strategies at higher difficulty levels. So, to enhance the performance of the algorithm, it wouldn't be redundant, to use reward functions. For example, If the agent gets closer to the opponent's territory you get a (+1) reward, if it remains in the same line, i.e. moving(, left or right) you get a (0) reward, going down will result in a (-)1 reward, placing a "crucial" wall will result in +3 reward, and your function is maximizing your rewards. These are just the thoughts and implementation of these strategies both can be beneficial or at least will not harm the existing progress.

Chapter 6

Bibliography

1. Browne, C. B., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games, 4*(1), 1-43.

2. Carmel, D., & Markovitch, S. (1996). Opponent modeling in multi-agent systems. In *Adaptation and Learning in Multi-Agent Systems* (pp. 40-52). Springer.

3. Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *International Conference on Computers and Games* (pp. 72–83). Springer.

4. Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education, Inc.

5. Glendenning, L. (2005). Mastering Quoridor. Bachelor Thesis, Department of Computer Science, The University of New Mexico.

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

7. Brenner, M. (2015). Artificial Intelligence for Quoridor Board Game.

8. Respall, V. M., Brown, J. A., & Aslam, H. (2018). Monte Carlo tree search for Quoridor. In *19th International Conference on Intelligent Games and Simulation, GAME-ON* (pp. 5-9).

9. Bezáková, I., Heliotis, J. E., & Strout, S. P. (2013, March). Board game strategies in introductory computer science. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 17-22).

10. Jose, C., Kulshrestha, S., Ling, C., Liu, X., & Moskowitz, B. Quoridor-AI.

11. Khachikyan, N., Hovakimyan, A., & Tigranyan, M. Monte Carlo method for option pricing. `https://github.com/NarekKh/Monte-Carlo-Method-for-Option-Pricing`