Quoridor, a board game AI

Iker Eizagirre

1 Introduction

Artificial intelligence for board games has been on the minds of people for a very long time now. In fact, over a hundred years ago Charles Babbage (1791-1871) was already theorizing about how to program his Analytical Engine, which was never actually built, to play the classical game Chess. In the second half of the twentieth century, the discussion of whether computers could actually beat master level humans or not was so heated that International master David Levy made a bet stating that no machine would be able to beat him in the coming ten years. In the present, even the best players have little to no chance of winning on popular games like Chess or Go against programs running on highly specialized hardware, and tournaments and rankings are held separately for each.

Quoridor is a two player board game designed in 1997 by Mirko Marchesi. Very much like the previously mentioned games, it is a perfect information sum zero game. This means that both players have all of the information available about the current state of the game, and any winning move for a player is a losing move for his opponent. The purpose of this paper is to see how to implement and AI capable of challenging players in Quoridor by using general board game techniques and a specialized Static Evaluation Function.

2 The game

A Quoridor game is composed by a 9x9 shared tile board, 1 pawn and 10 wooden pieces per player. The game starts with the players having their pawns in opposing sides of the board. Please refer to Figure 1. The first player to reach his opponents side wins.

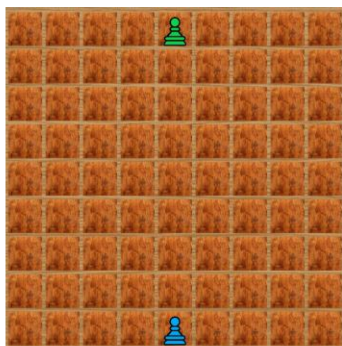

Figure 1 Initial setup for a game of Quoridor.

2.1 A turn

A turn in Quoridor consists of one of the players taking one of two actions. Either move her pawn, or place a wall.

A pawn can be moved by placing it in a cardinally adjacent tile of the board. If there is a wall between the current tile and the wished destination tile, the movement is illegal. If the opposing pawn is in the destination tile, then it is possible to jump over it and move further one extra tile. A final extra rule states that, if the opposing pawn is in the destination tile, but jumping over it is illegal because there is a wall behind it, then the movement can be done diagonally.

Refer to Figure 2 for example moves. Let's say that it is the blue players turn. Moving to the left is not possible because there is a wall in that direction. Right and down are perfectly legal moves. Since the opposing pawn is up, we could jump over it, but in this case there is a wall behind it so it is not possible. Instead, the blue player move to either greens right or left.

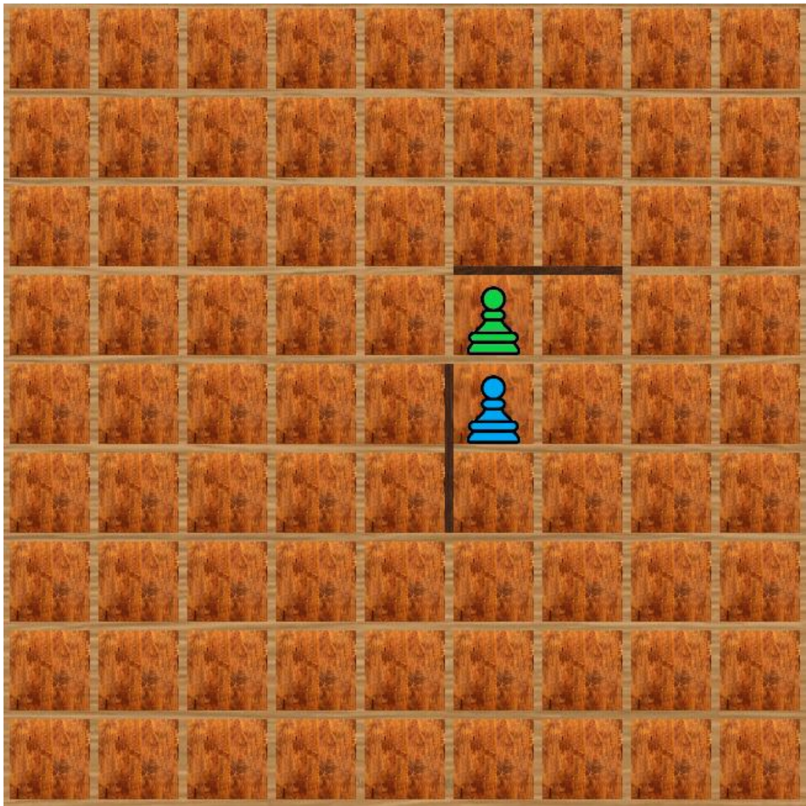

Figure 2 Example of the state of a game after a few moves

The other possible action to take in a turn is to place a wall. Each player has 10 wooden pieces to block the others path to victory. Walls can be placed anywhere in the board as long as two rules are not broken.

First, a wall cannot intersect another wall. If there is already a wall placed in the board,

a new one can be placed such that it touches the previous one, but can never be placed on top. Second, both players must always have a possible path to reach the end. If placing a wall means blocking the only possible path for a player to reach the last row, then that is an illegal move.

2.1 Finishing condition

Turns are played switching between players until one of them reaches the very last row in the opposite side. As soon as this happens the game ends and that player is the winner. Draws are not possible in Quoridor.

3 Speed

For the machine to be good at the game there are mainly two factors to take into consideration. One is the speed of the algorithm. What the program will be doing is simply search through a tree of possible moves and responses to moves and chooses the move with most potential.

Searching through all the possible actions until the finishing condition is met is unthinkable, as the amount of work to process increases exponentially. This means that a maximum search depth needs to be set. The deeper our program can search, the more accurate the result will be. Also, the faster our program is, the deeper it will manage to search in a reasonable amount of time. In conclusion, faster algorithms yield smarter moves.

This is where general algorithms for board games come into play. These algorithms focus on searching through the tree of possible moves as efficiently as possible.

3.1 Minimaxing

The minimax algorithm considers the potential value that doing every possible move in a board state yields, and then chooses the best move depending on the values of each move. The value for each move is again computed analyzing the possible responses. The recursion is stopped by calling the Static Evaluation Function to get an approximation of the current board state potential.

Depending on whose turn it is, the action chosen from the possible pool of moves is different, because we will try to maximize our score, but out opponent will try to minimize it. There is a variation of the minimax algorithm called negamax. It consists on negating the potential values on each iteration so that the code to handle both players turns is the same.

3.2 AB Prunning

Alpha-Beta pruning is a technique to avoid searching through some branches of the tree when they are not needed. It consist on creating a range of potential, limited by a minimum value where we will not allow moves that yield worse potential than the minimum, and a maximum value, where our opponent will not allow moves that yield more potential than the maximum.

As an example, let's say that the branch originating from the first possible move of our turn yielded us a potential of 5. If any of the possible responses to the second move that we are analyzing yields a potential of 4, there is no need to keep searching. Move 1 is already undoubtedly superior to move 2. The inverse happens when our opponent is analyzing responses to her move.

3.2 Zobrist hashing

When all the possible moves for a board state are getting analyzed, in Quoridor it is almost certain that sets of moves in different orders will reach to the same board state. In these cases, it is possible to avoid the extra work of duplicated tree searching by storing the potential of each board state we reach and checking if a board state has already been analyzed every time a move is done.

For this to work well we need to avoid collisions. Thus, it is essential that little changes in board result in different hash values, because the change to the board in a single turn is minimal. Zobrist hashing fulfills this purpose very well. A specific implementation for Quoridor would go like this:

The play board in Quoridor has 9x9 tiles, so we create an array of 81 elements. The situation of each tile is defined by 4 different elements that can happen separately. These are:

- Player 1 being on the tile
- Player 2 being on the tile
- A horizontal wall being on the tile
- A vertical wall being on the tile

In reality horizontal and vertical walls are not part of the tiles, and they go in separate grids, but since the sizes of the grids are also almost 9x9s they have been combined for the board state to fit the Zobrist arrangement. As there are 4 different binary options, there are a total of 16 combinations, so what is actually created is a 2D array of size 81*16.
Each of the positions of the array is filled with a random string of bits of a fixed size.

When we want to get the hash value of a board state, the random strings of bits corresponding to each tile state are XORed together.

4 Static Evaluation Function

When the tree search reaches the depth at which it has been set to stop, an approximation of the potential that board state has for the player needs to be computed. This is what the Static Evaluation Functions does. It is basically a sum of different values corresponding to various aspects of the game that make the current state to be good or bad.

For example, one of the factors we could look at in chess is how many pieces each player has left. Another piece of information to take into consideration is how many tiles of the board are controlled by each player. The sum of these two aspects gives an approximation of how good the board state is.

In the case of Quoridor, there are mainly two factors to take into consideration, distance to the end and amount of walls left.

4.1 Distance to the end

It refers to the minimum amount of steps the pawn would need to take in order to reach the end row avoiding the currently placed walls and without modifying the board in the process. This information by itself is meaningless, but when compared to the opponents distance it is the clearest data of how much better than him we are doing. It is a good idea to multiply the difference by a scalar, in order to make it possible for other factors to yield values that are smaller than what corresponds to a distance of 1.

The minimum distance can be computed by doing an A* search. The estimation for each tile would be the distance to the end in a straight line and with a weight of one.

4.1 Amount of walls left

Once again, the amount of walls left needs to be compared to the amount of walls the opponent has left. In this case, and after some experimentation, it seems that an appropriate potential is the difference squared.

Taking into account just the difference placing walls is considered beneficial as long as the enemies’ path length is stretched enough. In reality though, no matter how much shorter a players’ path is, if he runs out of walls sooner by far, then he is very likely to lose.

In the case were the actual value is the difference squared, the only situation in which running out of walls to place is not considered a really bad move, is when the opponent also has few walls left. In this situation, the board will be filled with walls, and it is likely that there are not that many good places to play walls left.

5 Conclusion

In order to be able to implement a competent Quoridor AI, two things are needed, speed and an accurate Static Evaluation Function. To get an efficient tree searching, general board game techniques such as minimaxing work very well with perfect information and sum zero two player games, and Quoridor is no exception.

As no popular game is too similar to Quoridor and the game itself does not have any type of inner scoring system, the factors to consider unto the Static Evaluation Function need to be experimented upon. But the difference in the shortest path for the players and the difference squared of the amount of walls left seems to work well enough.

6 References

Ian Millington. 2006. *Artificial Intelligence for Games*. 647-690. Morgan Kraufmann.
https://en.wikipedia.org/wiki/Computer_chess

Images

https://static.vecteezy.com/system/resources/previews/000/125/950/non_2x/vector-wood-texture.jpg
http://www.texturex.com/albums/Wood-Textures/hard%20wood%20texture%20floor%20plank%20smooth%20shine%20cherry%20wallpaper.jpg
http://icons.veryicon.com/ico/System/iOS7%20Minimal/Chess%20Pawn.ico
https://s-media-cache-ak0.pinimg.com/736x/e9/61/e8/e961e86c757550786edf1b5e5034ca45--dark-wood-background-dark-wood-texture.jpg