# Quoridor Agent

## AI agent for a complex board game

Guy Tsur
Introduction to Artificial Intelligence Course (67842)
Hebrew University of Jerusalem
Jerusalem, Israel
guytsur7@gmail.com

Yotam Segev
Introduction to Artificial Intelligence Course (67842)
Hebrew University of Jerusalem
Jerusalem, Israel
yotamseg@gmail.com

*Abstract*— **In this paper we consider several concepts for an AI agent playing the game Quoridor. The game has a high branching factor similar (and even greater) than chess. In the program we consider several methods of decreasing the branching factor and numerous heuristics for evaluating the value of a given game board (state). Lastly we consider a genetic algorithm for weighing the different components of a meta-heuristic, one that is composed of several simple and varied partial heuristics (parameters).**

***Index Terms*— AI, Quoridor, minimax, alpha-beta pruning, branching factor.**

## I. Introduction

The game of Quoridor is a child in the realm of board games, invented in 1997 by Gigamic. It is an unsolved and still mysterious game which holds many complexities and strategies, some already known and identified and others left for future investigation. Games like this one have held a place of honor in society since ancient times. The first games were played more than five millennia ago and predated literacy. Men have found the games intellectually challenging and the competition, a test of wits between adversaries.

With the rising dominance of computers in modern society a new challenge arose. The challenge of creating computer software capable of defeating human players in a game. For some games this challenge is possible and even more than that, the game itself can be solved completely (Tic-Tac-To and Checkers etc). Quoridor is an unsolved game with high complexity similar to chess.

Our goal in this project is to create an AI agent that will challenge a human player in the game. We will tackle this goal from different directions. It is important to note that the reactiveness, the time to play, of the agent is constrained by the human player experience and creates a tradeoff between quality of decision and time of decision.

## II. The Game of Quoridor

### *A. The Rules of Quoridor*

Quoridor is played on a 9X9 square grid board. Each player has a pawn on the board in opposite edges of the board. The goal of the game is for your pawn to reach the opponent's edge. Varieties of the game with more than two players exist but are not in the scope of this paper. Every turn the player has two options:

1. Jump: move a single square in the four directions. Movement is limited by the edges of the board and by walls. The opponent's pawn does not limit movement, rather it is hoped over.
2. Build a wall: walls are 2 squares long and are placed on the edges between squares either horizontally or vertically. A wall can't intersect another wall and can't block the opponent's ability to reach the opposing edge, the goal. Each player has an allowance of ten walls per game.

A simple calculation will show that a player can have $4\ jumps + 8 \cdot 8 \cdot 2\ walls = 132\ possible\ moves$. This is quite a bit of moves compared to most popular household games and goes to hint at the complexity of Quoridor.

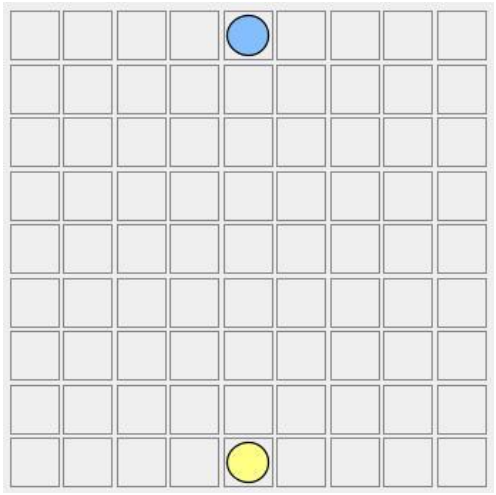

*Figure 1 – Quoridor game board*

### *B. The Complexity of the Game*

The complexity of the game is due to three aspects, lack of knowledge, number of possible moves and lack of performance indicators.

*1) Lack of knowledge:*

Quoridor is a rather young game. Research on it is very limited compared to classic games like chess or checkers (which has been solved completely to a draw in 2007 by Chinook). Thus, we could not find abundant information about strategies, opening moves and perspectives about the game. It goes without saying that our experience as players was also fairly limited and we are far from expert Quoridor players.

*2) Number of possible moves:*

The high number of possible moves allowed for each player each turn creates a game with too many moves to calculate completely. A calculation four moves deep can be composed of considering more than $3 * 10^8$ possibilities and that number grows fast when considering more moves. This puts a tradeoff on the agent planned, it must prioritize what to consider and what not or settle on a very shortsighted view of the game.

According to Mertens [4] two main parameters need to be calculated to consider this aspect of the game's complexity. The first, state space complexity is an upper bound on the number of possible board settings (states) of the game. It is calculated by a multiplication of the number of pawn configurations and wall configurations, disregarding the legality of the state.

$$(1)\ S_{pawns} = 64 \cdot 64 = 6480$$

$$(2)\ S_{walls} = \sum_{i=0}^{i=20} \prod_{j=0}^{j=i} (128 - 4i) = 6.16 \cdot 10^{38}$$

| Game | log(state-space) | log(game-tree) |
|---|---|---|
| Tic-tac-toe | 3 | 5 |
| Nine Men's Morris | 10 | 50 |
| Awari/Oware | 16 | 32 |
| Pentominoes | 12 | 18 |
| Connect Four | 14 | 21 |
| Checkers | 33 | 50 |
| Lines of Action | 24 | 56 |
| Othello | 28 | 58 |
| Backgammon | 20 | 144 |
| Quoridor | 42 | 162 |
| Chess | 46 | 123 |
| Xiangqi | 52 | 150 |
| Arimaa | 42 | 190 |
| Shogi | 71 | 226 |
| Connect6 | 172 | 140 |
| Go | 172 | 360 |

*Table 1 – board game complexities*

$$(3)\ State\ Space\ Complexity = S_{pawns} \cdot S_{walls} \cong 4 * 10^{42}$$

The second, game tree complexity is a summation of the number of states that can be considered when playing a full game. It is calculated by raising the average branching factor (possible moves) by the power of the average number of turns per game. In her paper, Glendenning [1] stated that the average number of turns is 91.1 and the average branching factor is 60.4.

$$(4)\ Game\ Tree\ Complexity = 60.4^{91.1} = 1.79 \cdot 10^{162}$$

A comparison to other known games shows that Quoridor is right up there next to chess, in that league of complexity brute force algorithms and prior knowledge will not be enough for an upstanding agent and deeper thought will be required to create such an agent for this difficult game.

*3) Lack of evaluation features:*

Several of the papers we've read attempting to consider evaluation features had trouble finding meaningful and relevant ones. All of them deal with your distance from the goal, the number of walls you have and other trivial and shallow considerations. If those papers would have seen spectacular success with their agents we would have been satisfied with these features but sadly they all regarded their agents as wanting. For comparison, Deep Blue, the famous program that defeated chess champion, Kasparov, consider over 8,000 of these [3].

## III. The Method

After our initial research we came to realize that a gradual approach is in order. We decided to start our work by creating a most basic depth limited minimax agent that uses a simple difference of path lengths as its evaluation function. From there we divided the challenge to three parts upon which we will expand. The evaluation function, the branching function and human like insights. Each of these problems we attempted to tackle using several different methods and the different combinations of these methods and their parameters gave birth to several agents.

## IV. The Basic Agent

The Quoridor game is a classic case for using the minimax decision rule to select the move that minimizes the possible loss for a worst case scenario. Minimax assumes that the opponent is an optimal player according to its evaluation function and thus selects the move whose outcome will allow the opponent the worst optimal move. The depth of the tree, in our case, is limited since the computation duration shouldn't hurt the human player experience. Thus, an agent considers the move it will make and the opponent will retaliate back and forth until a maximal depth has been reached.

Minimax is optimized using alpha-beta pruning. Alpha-beta pruning attempts to decrease the number of nodes explored. This is done by stopping the exploration of a node if there exists at least one other node which is proved to be better. This algorithm can be

further improved by use of history and killer heuristics. These heuristics deal with the order in which the nodes of the tree are expanded which drastically change the effect of alpha-beta pruning.

The basic evaluation function is a mere path difference function which returns the reciprocal of the difference in best path length to the goal. This heuristic has no consideration by itself for the durability of the path, the number of paths available and so, but those considerations can come into play if the depth of the minimax tree is adequate, an attribute which is hard to attain due to the high branching factor.

## V. Evaluation Functions

### *A. Basic - Path difference*

The basic agent heuristic, described in detail in chapter IV.

### *B. Random walker statistics*

The idea in this evaluation function is to replace the two pawns with random walkers, pawns the each turn pick a random direction to walk. Running several games with these random walkers and looking at the winning statistics can tell a lot about the state of the board. Obviously it is a witness to the agent's proximity to the goal but the uniqueness of this method lies in the randomness of the agents which gives testament to something broader than just the proximity, the number of available paths to the goal, the width (how hard are they to block) of the path and more. This evaluation function has two parameters that affect it, the number of games run and the forward orientation given to the agents. The first is a tradeoff between running time of the heuristic, which in turn imposes a stricter limit on the tree depth. The second is a tradeoff between the randomness of the agents which helps identify all possible paths and their width to the common sense of the game which states that a forward orientation is most likely.

This heuristic has two phases in which it is weak and one in which it is strong. The beginning of the game and its very end are the weak phases. The beginning of the game is rather indifferent to by this functions' perspective. The first few moves of the game far from determine a random game's outcome and thus the heuristic is rather stupid in suggesting an appropriate course of action at this phase. At the very end when one of the players is extremely close to winning the function again becomes quite indifferent to the different moves. All of the moves have a high probability of winning and the number of games played has to be exceedingly high to accurately predict which move is best. In the core phase of the game when both the players are in mid board and there are a lot of walls around the function is very strong and returns surprisingly good moves.

*Note – this is a high cost feature, is takes long to calculate, and it's contribution to the evaluation of the board isn't always profitable against deepening the search tree or using a larger branching factor.*

### *C. Weighted evaluation features*

We identified several more Evaluation Features in the game. The goal was to use the features together in order to get a better evaluation of the current state. Hopefully where one feature will give a wrong estimate (for example good path difference but one wall away from shutting down our path and sending our agent on a very long stride…) another Feature will be able to spot this (in the example, the Random Walker Statistics will be able to spot this).

The extra features we used (in addition to Random Walker Statistics and Path Difference):

- *Manhattan distance* – this is a common Heuristic in path searching problem. The idea is that in general we would like our pawn to get closer to the goal line, disregarding the walls on the way. In our game this is simply the position of the pawn on the Y axis.
- *Been there* – the idea behind this feature is that it is rarely a good idea to come back the way you came from. Also a vital feature in order to prevent an agent from moving back and forth between the same states.
- *Remaining walls* – a very simple feature, simply indicating that the remaining walls are an important resource.
- *Walls before you* – the number of walls in the remaining Y-axis area between the pawn and the target. Obviously not all of them are a real obstacle but an open board is better than a board with a dozen walls between you and the goal.

The next Challenge was to manage to combine all of these features into a single heuristic. We used a linear combinations of the different features.

$$(5)\ Eval(state) = \sum Weight_i \cdot Feature_i(state)$$

In order to try and find the best weight $(w_i)$ for each feature $(f_i)$, we applied a genetic algorithm.

### *D. Genetic Algorithm*

The concept behind Genetic algorithm is to try and mimic nature's way of natural selection and mutation. We create a "Population" of agents, let the "Fittest" survive and "Spawn" new and better fitting agents. It is an iterative process in which each iteration is called a *Generation*. Each generation has 3 stages:

i. *Mutation* – In order to advance and create "fitter" agents we alter stochastically a few of their parameters.
   - In our case, the "Population" was a single agent. From him we created his "Competitor" by randomly selecting a number of the weights and randomly multiplying them in a uniform distributed factor in the range [0.7-1.3].

ii. *Fitness Function* – this function evaluates who is the better agent, in order to select the best and let his "*Genes*" move on to the next generation, while the "*Genes*" of a badly mutated agent will disappear.
   - We chose fitness function was a simple run off between the original and the mutant agent. The fitness of the agent was simply the number of wins he was able to achieve against the other agent.

iii. *Crossover* – this part is analogous to "Reproduction". We take the fittest of our agents and make them create a better generation.
   - We used a simple crossover function, we took the weighted average of the parameters out of the two agents, with the winning percentage (our *Fitness Function)* as the weights.

$$(6)\ Weight_i = \sum_{AGENT=1}^{2} \frac{WINS_{AGENT}}{TOTAL\ MATCHES} * W_{i_{AGENT}}$$

## VI. Branching Function

### *A. Basic – All moves*

This branching function returns all the legal moves the player can make. According to Glendenning [1] the average number of moves per turn is 60.4. The maximum number of moves is 132. This branching function causes the search to be complete since it considers all possible moves but is quite heavy and puts a tight limitation on the tree depth.

### *B. Walls close to pawns*

This branching function is parameterized by a radius around each pawn. It returns the possible jumps and the possible walls in the given radius around each player. This is due to the assumption that most of the builds occur in proximity to the pawns. This function is incomplete and could cause the agent to miss a fantastic move that goes unconsidered. With this function, and a radius of 1, the upper bound on the branching factor is 40. Thus, the game tree complexity is reduced below that of backgammon.

$$(7)\ G = 40^{91.1} = 8.86 \cdot 10^{145}$$

If we assume a branching factor of 20 which seems quite appropriate based upon experiment we find that the game tree complexity reduces below that of chess.

$$(8)\ G = 20^{91.1} = 3.34 \cdot 10^{118}$$

It is interesting to note that assuming the same number of turns per game as Glendenning [1] we would have to drop consideration of walls completely in order to reach the game tree complexity of checkers, a solved game.

### *C. Walls close to pawns ordered by killer heuristic reasoning*

This branching function is similar to B with a distinction. While in B we made no consideration to the order in which the moves will be expanded, with this function we attempt to cause the "killer moves", those that will cause the alpha-beta pruning to trim the search most often will be checked early on and thus reduce the number of nodes expanded. We assumed that the killer moves will be the most prominent and common moves, move forward and build in front of the opponent. This idea is based on the concept of the killer heuristic [2]. In practice this had very little effect and was removed from our agent. The reason for its minor effect is that we usually considered walls in a radius of one around the players and thus the heuristic had no real effect on the order of moves checked.

### *D. Disregard opponent's build*

This is an adjustment to all of the functions above which simply decides to ignore the opponent's build moves and consider only its movement. This can be reasoned by the slim chance of a human opponent of being optimal by the agent's evaluation function and allows for the tree to be deepened substantially. It is possible to use this branching factor for your moves as well but obviously it will lead to very basic playing strategy. Using this adjustment with the all moves heuristic reduces the game tree complexity more than the walls close to pawns function.

$$(9)\ G = 60.4^{46} \cdot 4^{45} = 1.05 \cdot 10^{109}$$

Using this adjustment and the walls close to pawns function, with radius 1, reduces games tree complexity to a midpoint between chess and checkers.

$$(10)\ G = 20^{46} \cdot 3^{45} = 2.08 \cdot 10^{81}$$

## VII. Human Like Insights

### *A. Deepen depth when out of walls*

When using any branching function that considers building walls it is usually a far more prominent part of the branching factor than the movement. This can be used to our advantage if one or both of the players are out of walls. When there aren't any walls to build we can deepen the tree, an addition of one to the depth was chosen.

### *B. Sure thing state*

The different evaluation functions have strengths and weakness but there are also absolute truths, for example, if your opponent is out of walls and your shortest path to the goal is shorter than it's you should just walk to the goal and win. This is implemented in our agent as a "sure thing" state that is to be checked and acted upon regardless of which evaluation function is used.

### *C. Predetermined opening*

Chess, perhaps the classic and role model of all board games has countless "classic" openings, for Quoridor we found three. These openings, Reed, Shiller and Ala are a good starting choice for any player and for our agent as well.

Reed: the reed opening consists of placing two horizontal walls in the third line before the opponent with one space between them (the middle square).

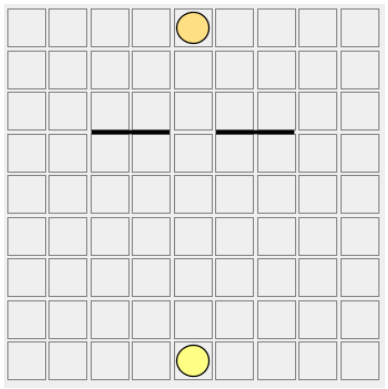

*Figure 2 – Reed opening*

Shiller: the pawns advance three steps each, the first player places a wall a vertical wall at its starting edge of the board in the row closest to the pawn. This opening gives the opponent two possible paths versus one path for yourself and is supposed to reduce the complexity of your game.

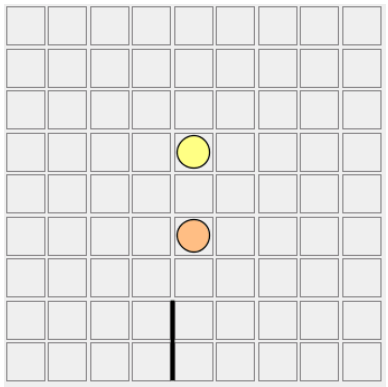

*Figure 3 – Shiller opening*

Ala: the pawns advance three steps each, the first player places a wall behind him and then another. The next two walls are built perpendicular to the wall behind the player towards the goal edge. This way the opponent has two paths open at the sides of the board. (Wasn't implemented in the agent)

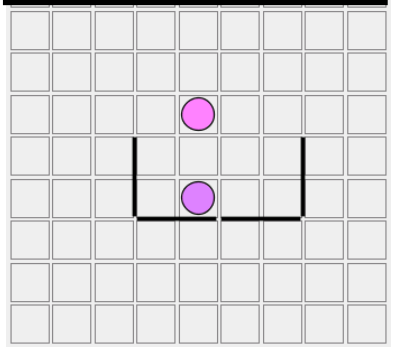

*Figure 5 – Ala opening*

### *D. Evaluation function tie resolution*

A dilemma we dealt with was how the agent should deal with a tie in the evaluation function's return value. At the beginning of our work we let it arbitrarily decide the outcome. Later we came to realize that this has tremendous effect on the game, especially in its early stages which are very hard to evaluate and concluded that a random choice is in order. Other approaches might be in order as well, consider having a set of popular moves which will have priority in a situation like these. These might be building a wall in front of the opponent or moving forward.

## VIII. Results

This chapter holds a special challenge, how to represent our results. A priori we thought about tables of game statistics, a ranking of the different agents we created. We began to collect the data and decided that it is quite irrelevant. Our goal was to create an AI agent that will challenge a human player and thus the only statistics we could have used we those of playing vs. human competent and that we sadly did not collect. We decided to consider the different agents we used and methods and expand upon their pros and cons, their strengths and weaknesses as perceived by us through rigorous hours of testing them against each other but especially against us and our fellow class mates.

All of the agent's below use the Sure Thing and Out of Walls Deepen Tree insights.

### *A. Basic agent:*

This agent uses the path difference evaluation function with the all moves branching function for itself and its opponent. The depth of the tree is the main parameter that improves it and is limited by the time allowed for each move, for a highly reactive game (less than 10 seconds reaction time) a depth of 2 is maximal.

Pros: this agent makes good, reasonable decisions. It takes into consideration your ability to block its main path and force it to use a secondary longer path.

Cons: suffers from the horizon effect, can't take into consideration game changing moves that are just beyond the depth it considers.

*B. Incomplete agent:*

This agent uses the path difference evaluation function with radius one close walls branching function for itself and the opponent. It can react well enough with a depth of 4.

Pros: this agent suffers less from the horizon effect. It is capable of making moves that are stronger and farseeing.

Cons: the agent can't take into consideration game changing wall building option like those cutting off one of a player's open paths.

*C. Incomplete disregard opponent build agent:*

This agent uses the path difference evaluation function with radius one close walls branching function for itself and movement only branching function for the opponent. It can react well enough with a depth of 6.

Pros: very deep sighted.

Cons: has little consideration for the durability of its chosen paths.

*D. Genetic 100*

This agent is comprised of five evaluation features and weights. It checks these features to a depth of two. It was created by running the genetic algorithm described in chapter V for 100 generations.

Pros: takes into consideration numerous heuristics that are supposed to cover for one another's weaknesses.

Cons: shallow, for the genetic algorithm to run enough generations it was necessary.

*E. Genetic 3*

Same as Genetic 100 but runs with depth 3.

Pros: runs deeper than Genetic 100.

Cons: weights optimized for depth 2.

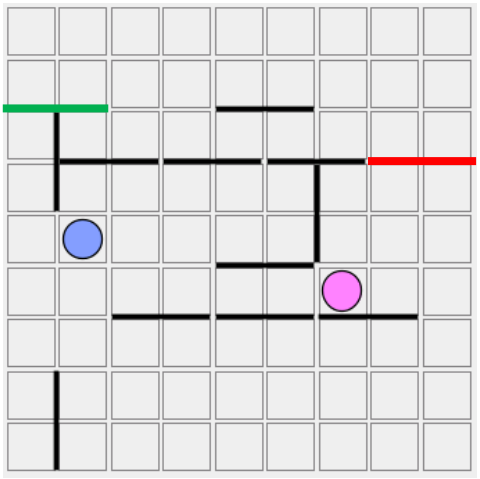

*Figure 6 – Incomplete agent downfall*

Notice that if the blue player is an incomplete agent he will not build the red wall since it is beyond it's scope and will get blocked by the green wall after it will waste several turns moving towards the left pass.

## IX. Summary

This project has taken us through a very interesting thought process. We began the process from the theory of AI as we learned in class, considered different representations of the problem, search methods and models. From there we moved on to the practical side of things, how to create an adequate agent for this difficult problem. We incorporated into the agent several human like insights and the close walls branching function which drastically improved its overall performance. Last we to identify several evaluation features (or use existing evaluation functions as such) and create a genetic algorithm to optimize their weights. Now, standing at the finish line of our project we feel is the time to go back to the theory of AI and incorporate new basic concepts into the agent with the rest of our work to improve it.

## X. Future Research

Future research on the game of Quoridor is highly recommended. We have observed numerous phenomenon which intrigued us and are quite wanting of attention.

Comparison between complete and incomplete searches: In a high branching environment should the completeness of the search be sacrificed for a deeper search? How to decide which nodes to expand and which to disregard? In this regard we recommend the research of quiescence search which attempts to identify hot and cold nodes, those that are worth exploring to a greater depth.

The horizon effect: is the name given to the undesirable limited sight of the minimax tree causing sometime tragic results when a move just beyond the scope of the tree has drastic results. This effect can be dealt with by using secondary search, meaning after a decision is made the node is rechecked to a greater depth to prevent a major turn of events just around the corner. This too can be used to improve a Quoridor agent especially with regard to the opponent placing a critical wall and turning the game around.

Build walls next to walls: on the same concept as considering only walls close to players it is appropriate to consider walls adjacent to already existing walls. An addition of this approach could compensate for the limited view caused by the incomplete search and recover some of the scope of considering all moves.

Evaluation features: though we have identified several such features we still feel that they are lacking. Perhaps Larry Shiller's upcoming Quoridor strategy book will help overcome the lack of knowledge and identify more and better features. It is evident from experience with different but similar games that the consideration of numerous features is a successful way to tackle such a game.

Different search mechanisms: it is possible that because of the high branching factor of Quoridor the optimal opponent assumption is not appropriate and different search methods more alike expectimax and negamax are in order.

## XI. Copyright Forms



## Acknowledgment

We would like to thank Martijn van Steenbergen for creating a wonderful kit for Quoridor AI programming.
Gigamic for this wonderful game.

## References

[1] Lisa Glendenning, "Mastering Quoridor",B.Sc. thesis, Computer Science, University of New Mexico, 2002

[2] Mark H.M. Winands, Erik C.D. van der Werf, H. Jaap van den Herik, and Jos W.H.M Uiterwijk, "The Relative History Heuristic", Department of Computer Science, Institute for Knowledge and Agent Technology, The Netherlands

[3] Russel S. and Norvig P. (1995a), Artificial Intelligence, a modern approach, Prentice Hall New Jersey

[4] P.J.C Mertens, A Quoridor-playing Agent, 2006

[5] Glenn Strong, The Minimax Algorithm, Trinity College Dublin, 2011

[6] Wikipedia: The free encyclopedia