Danmarks
Tekniske
Universitet

DTU

Board Game Assignment

Laser Chess

AUTHOR - GROUP 19

Eydís Sjöfn Kjærbo - s212943
Michal Marek Kossakowski - s220393
Bryndís Rósa Sigurpálsdóttir - s212423
Juliette Marie Victoire Vlieghe - s213705

Technical University of Denmark
Course 02180 - Introduction to AI

May 19, 2023

Contents

1 Introduction of Laser chess 1
1.1 Rules . . . 1
1.2 Analysis of the game . . . 1

2 Evaluation of our problem 2
2.1 State space . . . 2
2.2 Definition of our problem . . . 4

3 Implementation 4
3.1 States . . . 5
3.2 Algorithm for AI . . . 5
3.3 Evaluation function . . . 6
3.4 Adjustments of AI . . . 6

4 Future work 6

References 7

1 Introduction of Laser chess

Laser Chess is a two-player strategy game. The main goal of the game is to strike the opponent's queen while protecting your queen. When a queen is struck by a laser, the game is over and the player with the remaining queen wins. The official rules for the game include a king but we decided to change the king to a queen. Rules of the game can be found in the manual, which is cited [1].

1.1 Rules

At the beginning of each game, the player can choose the color of his tokens. The board is then set up as Ace [1] and the player with the blue tokens starts.

Each player starts with 13 tokens: one queen, two switches, two defenders, seven deflectors, and one laser. Each token has a role. The laser shoots a laser beam at the end of its player's turn and all tokens that are hit and don't reflect the beam are eliminated. The laser stays in its assigned corner and cannot be eliminated. All tokens except the laser have the task to defend their queen. Deflectors have three sides, two of them are non-mirrored but the other one has a mirror that reflects a laser beam 90 degrees. The defender has four sides, three non-mirrored and one mirrored reflecting the laser beam right back. The switch has two sides that are both mirrored and reflect a laser beam 90 degrees, because of this the switch cannot be eliminated. In the manual the switch can switch placement with an adjacent deflector or a defender, we decided to skip this part so we could put more focus on the AI. We also decided to skip the rule that says that if the same board arrangement appears for the third time in the same game, the player with the next move can declare a stalemate.

Players take turns, either rotating one of their tokens 90 degrees (the laser has to point to the table) or moving a token vertically, horizontally, or diagonally to a token-free tile (this excludes the laser) At the end of the player's turn, the laser is activated.

We decided to make an additional rule that was not mentioned in the official manual [1]. We decided that since each side of the queen is identical the player can't rotate his queen to skip his turn.

Some of the tiles are red or blue (instead of a helix) but only red tokens can be placed on a red tile and only blue tokens can be placed on blue tiles.

1.2 Analysis of the game

This game is a competitive game and a zero-sum game, where one of the players will end up winning and the other losing. The game is turn-based, but the two players take turns moving or rotating their tokens. Since both players can see all possible moves throughout the game and no move has an unexpected outcome, it is fully observable and deterministic.

This means that we could make an algorithm that uses tree search and searches the whole game tree for the best move. We could use a depth-first search for this project because there are many possible solutions and our game tree is quite broad but the depth can vary. This could be done by using a minimax algorithm including the alpha-beta pruning. On the other hand, we must examine the size of the state space and whether it is too large for us to search using this method. We may have to narrow the scope of our search. Maybe a more appropriate algorithm would be the Monte Carlo tree search where we can just train our algorithm to make the best decisions. But both of these algorithms are good for games that are turn-based, fully observable, and deterministic.

2 Evaluation of our problem

In this chapter, we want to get a better overview of our project by calculating our state space and defining the problem we are faced with. This will show us what exactly we can expect of our AI implementation.

2.1 State space

To get a better overview of the complexity of our project we would like to find the size of our state space. To do this we need to consider many things.

- Laser chess has a board with 78 tiles, 9 of these tiles are only accessible by the player with the red tokens and the other 9 are only accessible by the player with the blue tokens.
- We have two players and each player starts with 13 tokens, but could for instance lose 9 tokens before losing the queen and therefore losing the game. This means that before we lose the queen, we can have 576 different sets of tokens on the board, and we can have another 576 different sets of tokens on the board when one of the queens dies (same sets minus the queen).
- All tokens except the laser can move within the board, the laser has an assigned place on the table.
- All tokens except for the queen can rotate. A switch can have two positions on the same tile, a defender four, a deflector four, and a laser two.

We can use this information to make a function that calculates our state space.

Sign	Description	Quantity
m	All possible sets of tokens on the board before a queen dies	576
m_{end}	All possible sets of tokens on the board when a queen dies	1152
o	Number of deflectors on the board	0 to 7
e	Number of defenders on the board	0 to 2
l	Number of lasers on the board	2
i	Number of switches per player	2
t	Number of tokens on the table excluding the laser	5 to 24
$maxPT$	Upper limit on number of board arrangements	$\frac{78!}{(78-t)!}$
$minPT$	Lower limit on number of board arrangements	$\frac{69!}{(69-t)!}$
wq	With queen	
nq	Without queen	

We want to make our calculations a bit more simple and therefore use a lower limit and an upper limit. Our lower limit is the number of all possible board arrangements where the remaining tokens can be placed on 69 tiles (number of tiles minus half of the helix marked tiles). Our upper limit is the total number of board arrangements in which the remaining tokens can be placed on all tiles.

Lower limit:

$$\begin{aligned} &\sum_{n=1}^{m} 2l((4o_{Red})(4e_{Red})(2i_{Red}))((4o_{Blue})(4e_{Blue})(2i_{Blue}))minPT_{wq} \\ +2&\sum_{n=1}^{m_{end}} 2l((4o_{Red})(4e_{Red})(2i_{Red}))((4o_{Blue})(4e_{Blue})(2i_{Blue}))minPT_{nq} \\ &= 5.1 * 10^{48} \end{aligned} \tag{1}$$

Upper limit:

$$\begin{aligned} &\sum_{n=1}^{m} 2l((4o_{Red})(4e_{Red})(2i_{Red}))((4o_{Blue})(4e_{Blue})(2i_{Blue}))maxPT_{wq} \\ +2&\sum_{n=1}^{m_{end}} 2l((4o_{Red})(4e_{Red})(2i_{Red}))((4o_{Blue})(4e_{Blue})(2i_{Blue}))maxPT_{nq} \\ &= 1.7 * 10^{50} \end{aligned} \tag{2}$$

Note that if a player has no deflectors and/or no defenders, we want to let 4o and/or 4e equal to 1, instead of making that part equal to zero.

After careful calculations, we found that the state space lies between $5.1 * 10^{48}$ and $1.7 * 10^{50}$.

This implies that we cannot make an AI that can go through all of our state-space within a reasonable time limit before deciding their move. We need to decide how far down the search tree our AI should go or use a Monte Carlo algorithm.

2.2 Definition of our problem

In this section, we want to define our problem formally.

Our **initial state** (s_0) is as shown in figure 1. Either the AI starts the game or the human player depending on which color the human player chooses for his tokens.

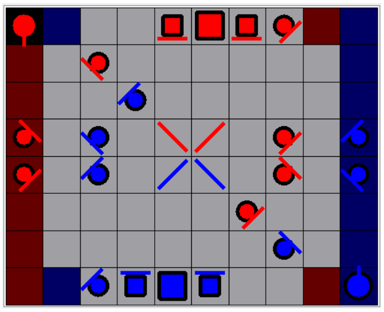

Figure 1: The initial state of the game

We have two **players**, distinguished by red tokens and blue tokens. The human player decides whether he would like to have the red tokens or the blue tokens. The function PLAYER(S) indicates which player has the move. For each move player can choose between three types of **actions**, that is move or rotate one of his tokens or rotate his laser. Each action ends with the player's laser being fired. The function Actions(s), returns a set of all possible moves. The **result(s, a)** of an action can be that nothing happens except that the state has changed slightly, or that one of the players has lost a token, potentially ending the game. We would like to limit the search by depth and therefore we need an evaluation function **Eval(s,p)** and a **CutOff-Test(s)**. These functions are used instead of a Terminal-Test(s) and Utility(s,p) function since we won't evaluate just the terminal state. The evaluation function calculates the game's predicted utility from a particular position.[2] This function will be explained in details in subsection 3.3. The cutoff test determines when to use the Eval function. We decided to use a depth limit and therefore we will use the Eval function when we have reached a certain depth.

3 Implementation

In this chapter, we will describe how our AI is implemented, that is, how our states are represented, how our algorithm works, how our algorithm evaluates each state, and adjustments of our AI.

3.1 States

A game tree is used to represent states and moves, with the root node indicating the game's current state. Every move that might be made at that moment will be the children of that node, where each node in the tree represents a conceivable game scenario. The Alpha Beta Pruning procedure is used to reduce the number of nodes or branches examined by the Minimax recursive Algorithm, which stores each player's probable action in an array. To develop the AI for this game, it was necessary to represent the game state as the placement and rotation of the game pieces. There is no requirement for a belief state in the game state since it is a fully observable game.

3.2 Algorithm for AI

As discussed before we could use tree search with depth-first search strategy, for example, minimax algorithm, and even include alpha-beta pruning to make the game tree smaller. Since our state space is very large, it would be important to limit the search by depth. We could also use the Monte Carlo algorithm to make our algorithm learn to make the best decision by experience. We chose to use the minimax algorithm including alpha-beta pruning and limit the search by depth.

Our algorithm is similar to the algorithm in figure 5.7 in [2] but uses an evaluation function and a cut-off test. Our algorithm starts by checking if it has reached the depth limit and if so it evaluates the state it is in. Otherwise, it checks if it should maximize the value.

- If so it gets all possible actions and goes through every action and calls the minimax method with this new state and marks that it should next minimize the value and increases the depth. By calling recursively the minimax method it will eventually return an evaluation value. Then the algorithm undoes the last move and finds if the value is the maximum evaluation value and sets the α as the maximum evaluation value. If α is greater than β we can stop going through actions.

- Otherwise it goes through similar steps as above but calls minimax and marks that it should maximize the value in next turn and finds the minimum evaluation value.

Our algorithm will finally return the best action if it has checked all leaves at this depth limit or returns the evaluation for the leaf it is checking.

This algorithm is suitable for our game since we have a very broad and deep game state and by doing a depth-first search, limiting the search by the depth, and using alpha-beta pruning, we make our algorithm faster. Our algorithm suits our game since we can map all possible moves because the game is deterministic and fully observable. It would be best if we also used the Monte Carlo algorithm and trained our AI since simple evaluation functions can be easily fooled. [3]

3.3 Evaluation function

The evaluation function is similar to the one from [4], which is based on the number of pieces and their proximity to the queen. Because the lasers and switches are constantly there, the values cancel each other out. The evaluation function used by the implemented AI returns the Manhattan distance to the queen of the same color. Table 1 shows how the Manhattan distance is calculated between different tokens.

Token	Value
Defender	10000 divided by the Manhattan distance to the Queen.
Deflector	75000; + 5000 if it sends the laser into the playing field.

Table 1: Evaluation function used by the AI

3.4 Adjustments of AI

The parameters in our algorithm that we could have adjusted are the depth limit and the evaluation function. After a limited number of games played by us versus the AI, we can conclude that the AI is capable of playing the game at a solid amateur level.

4 Future work

Unfortunately, we were unable to put our game to the test against someone with prior laser game expertise or against someone who has never played the game.

In the future, it would be fascinating to add other types of AI algorithms, search depths, heuristics, and evaluation functions and have the methods compete against each other and human players. Also, create benchmarks that compare the outcomes of the various versions and their speeds. Furthermore, we'd like to create a benchmark to see if it makes a difference whether or not a player starts the game.

References

[1] T. Fun, “Laser chess, instruction manual.” `https://www.thinkfun.com/wp-content/uploads/2017/10/LaserCh-1034-Instructions.pdf`. Accessed: 2-3-2022.

[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010.

[3] P. Muens, “Minimax and monte carlo tree search.” `https://philippmuens.com/minimax-and-mcts`. Accessed: 21-3-2022.

[4] J. Nijssen and J. Uiterwijk, “Using intelligent search techniques to play the game khet,” 10 2009.