Analysis

Introduction

For my computer science project, I will be making a grid-based version of the game “Draughts (Checkers)”. I intend to create artificial intelligence for use in a turn-by-turn game. As I will most likely be working with an array, I also intend to create a save and load feature for the game. My user is Matthew Richmond, a keen player of strategy games and avid mathematician. The game is well known and there are existing programs that simulate checkers, I can compare these to see what makes a checkers program refined. The main challenge is implementing different algorithms for the artificial intelligence.

What is Draughts?

Draughts is an 8x8 board game where there are two players and each player has 12 pieces. The player can move a piece into a diagonally adjacent square in the direction of the opponent. When it encounters an opponent’s piece, it can remove it from play by moving to the tile after it. When a player’s counter reaches the end of the board, that counter becomes a ‘King’, a promoted counter, which can move backwards. As you can see below, the only tiles that all the counters can occupy are the grey-coloured ones.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ❶ |  | ❷ |  | ❸ |  | ❹ |
| ❺ |  | ❻ |  | ❼ |  | ❽ |  |
|  | ❾ |  | ❿ |  | ⓫ |  | ⓬ |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| ⑨ |  | ⑩ |  | ⑪ |  | ⑫ |  |
|  | ⑤ |  | ⑥ |  | ⑦ |  | ⑧ |
| ① |  | ② |  | ③ |  | ④ |  |

Figure 1: Board in starting state

Abstraction of the problem:

There are five possible states that a tile on the board can be in: no counter, player 1’s counter, player 2’s counter, player 1’s promoted counter, and player 2’s promoted counter. You can use this to translate Draughts into 8x8 grid filled with these states. The grid has to follow these axioms:

1. A counter can only move into an empty square.
2. A counter can only move 1 or 2 spaces diagonally.
3. A counter becomes a promoted counter when it reaches the other side of the board, and retains all the properties of the normal counter.
4. Normal counters can only move forwards, towards the opponent’s side of the board.
5. Promoted counters can also move backwards.
6. A player can win by eliminating the entire opponent’s counters, or leaving no legal moves for the opponent.
7. A draw occurs when both players cannot force a win.

Artificial Intelligence

The Minimax Decision Rule  
  
To create artificial intelligence, I needed to research about different ways of representing and implementing a solution. Firstly, I found a way to differentiate between moves that are better or worse by checking what it would do to the resultant amount of checkers of checkers on the board. A move leading to a decrease in the amount of opponent’s checkers it would have a higher score than one that would decrease the computer’s.

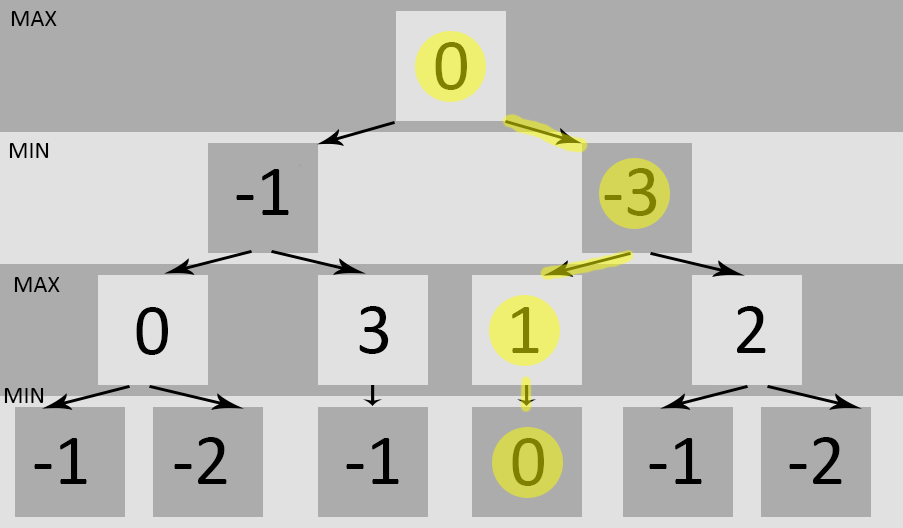
This is when I came across the minimax decision rule, a rule that helps you pick the best moves for the next few turns. This is done by taking a scenario where the amount of moves available is low, then listing out all the possible moves after that. By creating a decision tree, in which you label the ‘score’ of the move, you can pick the best score as the last move and play out those moves. Each branch leads to the opponents move and so on, which means you are assuming that the opponent is playing perfectly (using the same strategy).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ❶ |  | ❷ |  | ❸ |  | ❹ |
| ❺ |  | ❻ |  | ❼ |  | ❽ |  |
|  | ⑪ |  |  |  | ⓫ |  | ⓬ |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | ⑧ |  |  |
| ⑨ |  | ⑩ |  |  |  |  |  |
|  | ⑤ |  | ⑥ |  | ⑦ |  |  |
| ① |  | ② |  | ③ |  | ④ |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ❶ |  | ❷ |  | ❸ |  | ❹ |
| ❺ |  | ❻ |  | ❼ |  | ❽ |  |
|  | ❾ |  |  |  |  |  | ⓬ |
|  |  |  |  | ⑪ |  |  |  |
|  |  |  |  |  | ⓫ |  | ⑫ |
| ❿ |  | ⑩ |  | ⑦ |  |  |  |
|  | ⑤ |  | ⑥ |  |  |  | ⑧ |
| ① |  | ② |  | ③ |  | ④ |  |

Figure 2: Score = +1 Figure 3: Score = -1

Figure 1 and 2 show two examples of the scoring system, where the score is determined by the difference in the number of counters each player has eliminated. One board has a score of +1 and the other has a score of -1, if this were in a decision tree the move that has the outcome of +1 would be favoured over the other outcome. In figure 1 and 2, normal counters have a value of 1 and promoted counters have a value of 2.

  
Figure 4: Minimax Tree

The Minimax tree in figure 3 shows that every level it alternates maximising and minimising the player, the highest value at the end of the tree is what the artificial intelligence wants to base their next move on. Maximising the player is choosing the board that has the highest value and minimising is choosing the board that has the lowest value. Minimizing the player is done when you are assuming the other player is playing perfectly.

Objectives

{after discussion with the user}

1. The program must have a grid-based tile system to represent the counters on the draughts board.
   1. The tile system must be able to handle all types of counters commonly used in draughts.
   2. The tile system must be able to update in real time, such that a move can be shown graphically.
   3. The player must be able select two tiles on the grid, to allow the player to make a move.
2. The program must be able check if a move is legal.
3. The program must conduct the game turn-by-turn, so that the player can play against the artificial intelligence.
   1. The artificial intelligence must be able to look a certain amount of moves ahead, to decide the best move based on the parameters it has been given.
   2. The artificial intelligence must be able to check if a move is legal or not.
4. The program must have artificial intelligence and the player must be able to change the difficulty of the artificial intelligence.
   1. The artificial intelligence must have three difficulties: easy, intermediate, and hard.
   2. The artificial intelligence must use the minimax algorithm to choose the next move.
   3. (Optional): The algorithm can be made more efficient by using alpha-beta pruning
5. The program must have save and load functionality.
   1. The save function must be able to save the state of all tiles on the grid and save the difficulty of the game.
   2. The load function must be able to load the state of all the tiles to the grid and maintain the difficulty that is specified in the save file.

References

1. <https://en.wikipedia.org/wiki/Draughts>
2. <http://www.flyingmachinestudios.com/programming/minimax/>
3. Fig. 1,2,3,4: Rahul Yadav