Analysis

Introduction

For my computer science project, I will be making a grid-based version of the game “Draughts (Checkers)”. I will be working with a database, to create a save-game function, create artificial intelligence and a turn-by-turn game. My user is Matthew Richmond, a keen player of strategy games and avid mathematician. The game is well known and there are many programs that simulate checkers available, which I can compare with and critique. The main challenge is creating and 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 |  |

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. Normal counters can capture multiple pieces by having multiple successive counter captures using the same counter.
7. A player can win by eliminating the entire opponent’s counters, or leaving no legal moves for the opponent.
8. A draw occurs when both players cannot force a win.

Artificial Intelligence

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 amount of checkers. 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).



Here are two examples of the scoring system, where the score is determined by the difference in the amount of counters each player has eliminated. As you can see, one board has a score of +1 and the other has a score of -1, is this were in a decision tree the move that has the outcome of +1 would be favoured over the other outcome.

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

Score: +1, where score = P1 (White) – P2(Black).

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

Score: -1

Alpha-Beta pruning

When using the Minimax decision rule, the time to make a decision N moves ahead increases at O(bN), where b is the branching factor, (the average moves per turn). Alpha-beta pruning aims to decrease that by ruling out whole branches in the decision tree, resulting in O(bd/2).

Objectives

1. The program must have a grid-based tile system to simulate the movement and positions of the pieces of the draughts board.
2. The program must have the ability to play the game turn-by-turn, so that you can two players on one computer and one player playing against the artificial intelligence.
3. The program must have artificial intelligence, for the 1 player mode, and the player must be able to change the difficulty of the artificial intelligence.
4. The program must have save-game functionality that allows the user to save the game to a file. The user must also be able to load any compatible file to play.
5. The user must be able to train themselves using my user guide.

References

1. <https://en.wikipedia.org/wiki/Draughts>
2. <https://en.wikipedia.org/wiki/Minimax>
3. <https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning>