COMP1130 Assignment 3

Manindra de Mel, u7156805

Tutor: James, LAB: Tuesday 13:00 - 15:00

Introduction

Assignment 3 explores several complex algorithms and data structures and applies them to a board game. The board game in the provided assignment is Fanorana which consists of a 9×5 board and two opposing players, which have to completely eliminate the enemy pieces in order to win. Although the game itself has been written, an implementation for Artificial intelligence is required by the assignment, as such this report will explore the minimax algorithm and how it has been applied to Fanorana, as well as the possible shortcomings of the algorithm.

The Heuristic Function

Before our minimax algorithm can be implemented, we first need a method to evaluate the board (`GameState`), and assign a board a rank that can be compared to other boards during the Artificial Intelligence's search for the best board in it's favour. There are a seemingly infinite methods of ranking a board. A simple implementation would be the difference in pieces between the players. However, more complex heuristics would encourage the Artificial intelligence to play in a manner that we think may be optimal for it. The heuristic function would play the biggest factor in the performance of an artificial intelligence.

In the current implementation, the heuristic function rewards the Artificial intelligence to take more advantageous positions during the game. Moreover, these advantageous positions are any positions that aren't on the border of the board, where the piece has the least restricted movement. However, it is important to find a balance between taking advantageous positions and also killing the enemy. For example, if the Artificial intelligence is rewarded too much for positioning, it prioritises finding an optimal position over killing the enemy. Therefore, for each piece the Artificial Intelligence had in an advantageous position, It was only rewarded an extra 1 point for having a piece in that location. Similarly, if the enemy also had a piece in an advantageous position, it would also be punished (-1 points). As such, the heuristic function as of now consists of the difference between the pieces on the board with the consideration of positioning.

The Minimax algorithm

With the heuristic function implemented, we can then move onto some other helper functions that minimax will require. First a method is required to find all the possible next board states will need to be created, that is `GameState -> [GameStates]`. Since the game itself has been implemented in `Fanorana.hs`, we can use their functionality to create the next board state given a move (`applyMove`) and then apply it to all the moves possible for the Artificial Intelligence's moves to create a list of all the possible next game states.

Finally, with our helper functions created, we can implement minimax. Traditionally, minimax, as the name suggests, maximises one player's moves whilst minimising the other player's moves. Minimax does perform this evaluation in a tree like structure, following down to a certain depth in order to attain the best possible path for the maximising player. However, the traditional implementation of Minimax uses a switch to switch between the maximising player and minimising player, since the games the algorithm are typically applied to, are games without consecutive moves for one player. However, this is not true in Fanorana, since a player can chain a series of moves as long as they remain to move in a unique direction and as such a possible path for a series of board states may look like `MaxPlayer -> MaxPlayer -> MinPlayer -> MaxPlayer -> ...` compared to the traditional `MaxPlayer -> MinPlayer -> MaxPlayer -> ...`. Additionally, if the Artificial intelligence finds a path where it wins, it's rewarded with $100,000$ points, so the Artificial intelligence chooses that path and ends the game as quickly as possible. Similarly, if the Artificial intelligence discovers that the opponent may win, it is punished with $-100,000$ points to deter the Artificial Intelligence from travelling down that path. Finally, we can simply apply our heuristic function to each of the possible next game states and then pass the moves from the bottom of the tree to the top of the tree to find the next best move for the Artificial Intelligence.

Testing

Comprehensive testing was done on the various functions created for the Artificial Intelligence. This testing included both black box testing and white box testing. Once I first defined the various helper functions I required for my Artificial intelligence, I wrote various black box tests for the envisioned outputs of my functions. However, as my workspace and program grew, so did my various functionalities and as such white box testing was also added to make sure all the functions were performing properly. Additionally, for the minimax function, I often created a tree manually and then manually computed the heuristics for all the possible game states. I only performed such a manual computation on game states with very few next game states or only calling minimax at a small depth, so that the manual computation was not a laborious task. This was a very successful method of testing as it not only tested the minimax's function but also simultaneously tested if the heuristic functions and the function which generates the next game states were also working properly.

Reflection

Alpha-Beta Pruning

Alpha-beta pruning is an optimisation that can be added to the Minimax algorithm to decrease the time to calculate the best move and as such increase the amount of time for the algorithm to search through deeper possible board game states. Unfortunately, alpha-beta pruning was not included for the current Artificial intelligence present within the assessment as once added, it seemed the performance of the Artificial intelligence decreased. Due to this, I reverted the Artificial intelligence to just use the minimax algorithm. However, I believe given a bit more time, I would be able to incorporate Alpha-beta pruning into my program.

Exploring different heuristic methods

Exploring different heuristic functions is another venture that I wished I took. As mentioned before, there are a seemingly infinite ways to write a heuristic function for the program. An optimal heuristic function can make a significant difference between the performance of the Artificial intelligence . As such, maybe writing a heuristic function that encourages the program to player more aggressively or defensively could all be play styles worth exploring given more time.

Final remarks

Although the minimax algorithm could've been optimised with both Alpha-beta pruning and a different heuristic function. I still believe that the current implementation suffices at beating most avid human players. The program rarely crashes and has been tested thoroughly, so it makes no illegal or suboptimal moves. Overall, the implementation satisfies the assignment requirements and plays against its opponents well.