Lab 1 – Applying α-β Pruning Minimax to Othello

Group: 4, Falkenström, Maximilian, Forsberg, Linus

INTRODUCTION

We were tasked with implementing an intelligent agent for the board game Othello, using the minimax algorithm. The algorithm was implemented in Java/Java FX using the IntelliJ Idea development environment. For solving the assignment we took advantage of the accompanying framework Othello-FX, as we wanted to spend as much time as possible on the actual implementation of the algorithm instead of spending time designing a user interface. Taking this route meant we had to spend some time reading the documentation for the framework, studying how it works, and what methods we could use to solve the task.

IMPLEMENTATION

The implementation of our algorithm happened incrementally. We started off by building a basic version of minimax, without using any pruning or cutoff. Only once we had a working function did we work to add the pruning and cutoff criteria. This way we avoided the trap of building the entire algorithm, realizing it didn't work, and being unaware as to what part of the algorithm was acting up.

The cutoff is made when the time limit of 5 seconds is exceeded, or when the search depth is greater than the depth limit and at least 35% of the time limit has passed. This check is made using the *cutOffTest(...)* function that comes with OthelloFX. The way it applies the utility value is by taking the amount of bricks of our agents color and subtracting that by the amount of bricks of the opponents color. This way we find out how many more or fewer bricks our agent would have compared to it's opponent at the game state that the algorithm is at.

One challenge we had that didn't exactly have to do with the algorithm itself was to figure out how OthelloFX works and how we could use it. Once we had figured out how to work with OthelloFX the main challenge was getting a first version to work. Adding the pruning functionality and cutoff criteria was fairly easy once the algorithm was working.

We followed the pseudo-code implementation of the minimax-algorithm provided in the slides from the Games and Search methods lecture. In that pseudo code only the utility value was returned from the min and max functions and then the search algorithm knew what node to take based off of that. Implementing this in our case was a bit harder, we needed the search function (getMove) to know what move would lead to the utility value it had gotten. We opted to go for an implementation where the getMove function has much of the same functionality as the maxValue-function and that means it knows the possible moves as well as the utility value. In the conclusion chapter we elaborate on what we would have done differently if we were to redo this.

CONCLUSION AND FUTURE WORK

In this assignment we learned how to implement the minimax algorithm with alpha beta pruning for the Othello game. We have learned different ways to think about how an intelligent agent works and how it evaluates various possible moves. Other domains that the minimax algorithm could be used in are mostly non stochastic games such as Chess, Go, Checkers and Tic-Tac-Toe.

Things that could have been done differently in this assignment is for example, spending a bit more time on reading the documentation for the Othello-FX framework and studying the methods before we started coding. That would have saved us a bit of time in the end.
Another thing that could have been done differently is creating a custom data holder alternatively use an external library such as the java tuples library, to be able to return both the utility value and MoveWrapper. That would have made the getMove method a bit cleaner and we would not have to reuse the same code from the maxValue method in there.

Number of words: 655