Softwarepraktikum SS 2022

Assignment 4 Report

Group 3

Jamie Anike Heikrodt	394705	anike.heikrodt@rwth-aachen.de
Thomas Pollert	406215	thomas.pollert@rwth-aachen.de
Jascha Austermann	422571	jascha.auster@rwth-aachen.de
Silyu Li	402523	silyu.li@rwth-aachen.de

Contents

0.1 Who did what?

- Task 1: Thomas, Silyu
- Task 2: Thomas
- Task 3: Jamie
- Task 4: Jascha

Exercise 1

Our implemtatio of the move finding algorithms works with a method called **moveEnumeration()** which returns an ArrayList of all possible moves. We wrote another function that sorts these moves according to the rating of the board they lead too. We did this by writing a basic Comparator and use it with Collections.Sort(), which is some mergesort implementation. This improved our calculations quite well as you can see in our benchmarking results part.

Exercise 2

This part of this weeks assignment deals with benchmarking. We evaluated our different move- finding algorithms (MiniMax, Alpha beta without sort, Alpha beta with sort) by measouring the time each algorithm takes to find a move and the amount of states each algorithm needs to calculate for thoses moves.// From those values, the computation time per state for each algorithm can be estimated.
The efficiency of the algorithms in terms of: **time the algorithms takes to find a certain move** and in terms of **the amount of states the algorithms need to calculate** , can be compared. The following part will view the data we obtained and present the conclusion we could make from that.

2.1 results

Over ten different maps for a fixed depth of three, our ReversiAi needed to calculate 865 different moves. We let each algrithm calculate a move everytime one is requested and we return one of them after.
Our Minimax implementation calculated **30169342** states in **548,972 s**. This leeds to a calculation time per state of **54 ms**.
Our Alpha-beta implementation with move ssorting turned off calculated **7775122** states in **162,778 s**. This leeds to a calculation time per state of **47 ms**.
Our Alpha-beta implementation with move ssorting turned off calculated **4334397** states in **93,694 s**. This leeds to a calculation time per state of **46 ms**. If youre interested in the data, here is an Excel-sheet with everything we measured. The columns(left to right) stand for the time minimax, alphabeta withouto search and alpha-beta with search take to calculate the same move. For each map theres one table. At the bottom of the table you can find the amount of states the different algorithms calculated for the game on the map.
`https://studentsrwthaachende-my.sharepoint.com/:x:/g/personal/as7mqpuzewaqwwwd_students_rwth-aachen_de/ETvO3bDTUrBItZIshH5kEN0BdQo0US77UvfagCuIrX5ApA?e=TwnPqz`

Exercise 3

Our implementation of Minimax and Alpha-Beta both basically use Depth-First Search. Therefore we implemented iterative deepening for both of them and use it to handle time limits.
We use a for loop for iterating the depth, starting with 1 and going up to the maximum given depth. After each calculation, we check whether there is still enough time for the next calculation as described below.
In the Minimax and Alpha-Beta Methods we also have a safety measure. Before each expansion of a node we check whether we are only 200ms away from the time limit. If so, we set a global variable abortSearch to true, leading to no further expansions. Iterative deepening will then use the result of the last complete calculation so we do not compare any heuristics from different levels.

3.1 Minimax

For calculating level $i+1$ for $i > 0$ we remember the time that it took to calculate Minimax to depth i. We estimate the time that it took to calculate level i by subtracting the calculation time for depth $i-1$ from the calculation time for depth i. We then estimate the time for the calculation of depth $i+1$ by taking the calculation time for depth i and adding the time for level i times a branching factor. For the branching factor of one game state we use the number of moves that are possible on this game state. At first, we used the maximum branching factor that we had in the calculation of depth i but that turned out to be too pessimistic. We now use the average branching factor and that works fine.

3.2 Alpha-Beta

The basic idea is the same as for Minimax. However because of the pruning, we cannot take the same branching factor. We decided to use the number of states in level i divided by the number of states in level $i+1$ and that turned out to work well.