IAI Project 1 - Othello

Group 16
19 March 2021

Sjoerd Hilbert Pijpers sjpi@itu.dk
Kasper Dahl Andersen kdaa@itu.dk

Search algorithm

Our alpha-beta search algorithm is specified in the file OthelloAI16.java. The decideMove method returns the best move to make for the given game state. We create an Arraylist with the possible legal moves for the specific turn and recursively calculate the scores for each of them (unless pruned) with the maxValue and minValue methods. We keep track of the best v-value and corresponding position (bestP), which in the end is where the token is inserted. If there are no available moves, then it returns a new Position(-1,-1), which will change the player in the game. The different plays are "tried out" on a temporary Game state, which copies the current Game state, and then continues for a certain amount of moves. This provides our AI with the possibility of testing out several plays while not altering the current Game state. A terminal test is made initially on every recursion to figure out if there are more legal moves or if our cut-off function has been reached - more on this later.

A move-method was added for simulating a move by copying the gamestate at that point in the game where the method is called and inserting a token at the respective position in the game. This method is called inside the maxValue and minValue methods.

A Timer class was added to print the runtime for calculating the best move (see Timer.java), in order to see if the algorithm responds fast enough.

Alpha-beta pruning

To eliminate unnecessary searches through the search tree that would not improve the best possible move and increase the time spent on calculating this we have implemented Alpha-beta pruning. Two variables alpha and beta are initially initialized as negative and positive infinity integers. These methods are passed along on every recursive call of our maxValue and minValue methods, which helps keep track of the best and worst values for each game state tested and pruning, or avoiding to test those which would not decrease or increase the two variables respectively.

Evaluation function

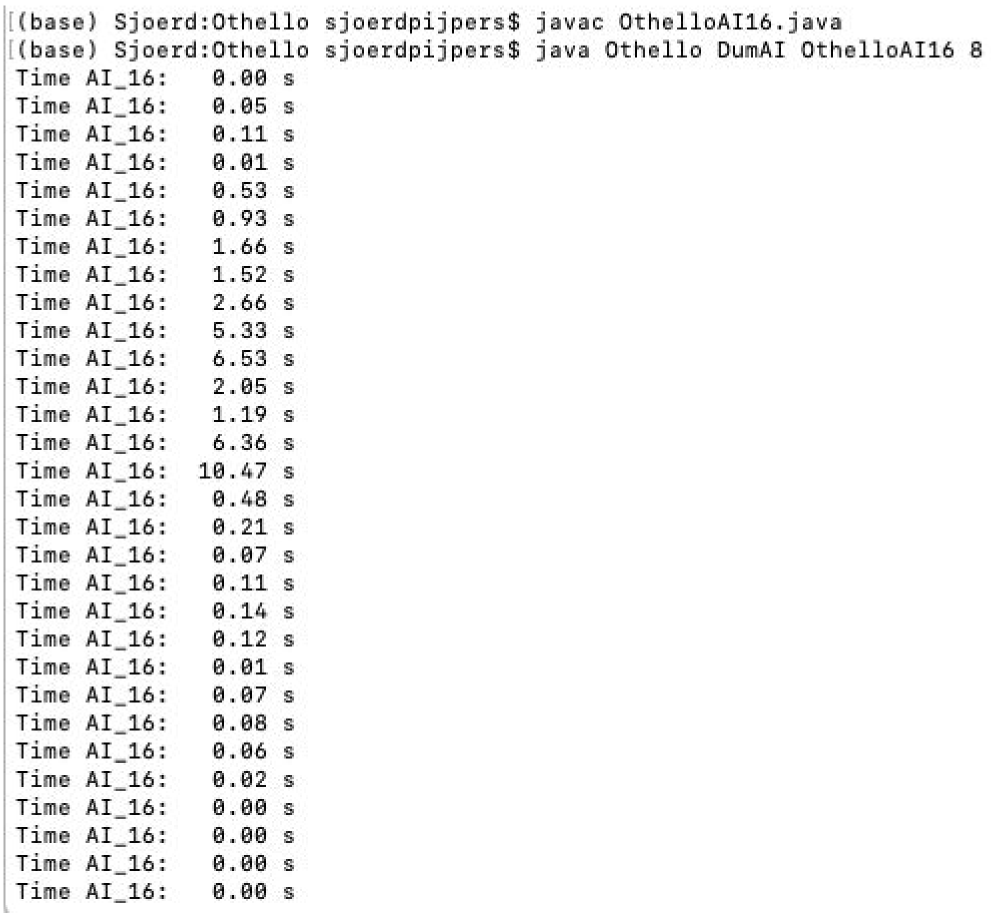
Our evaluation function is a combination of three heuristic functions[1]: coin parity, mobility and captured corners. It weighs each of these on a scale from -100 to 100 and then adds

[1] https://kartikkukreja.wordpress.com/2013/03/30/heuristic-function-for-reversiothello/

them up to give the final score. The coin parity function looks at the amount of black and white tokens on the board. The mobility function checks the amount of legal moves black and white have. The captured corners function checks how many corners are captured by both black and white. Corners have a high value compared to other positions because they cannot be flipped anymore once captured.

Cut-off function

We have implemented a cut-off function in order to cut off the search tree at a certain depth, by introducing a variable called depth set to 5 that is decremented every time maxValue and minValue methods are called inside the maxValue and minValue methods. This means that the AI evaluates 3 rounds or 5 ply's in advance. When it reaches zero, the Terminal-test becomes true meaning that a leaf is reached in the tree, and the evaluation function is calculated and returned. The cut-off function makes it possible for AI to calculate a good move within reasonable time (below 10 seconds). On the computer we used for testing, the time spent on calculating and performing a move almost never reaches more than 10 seconds, but a few moves were getting close to that region. Below is a picture of the times we measured for a game on the 8x8 board.



```
(base) Sjoerd:Othello sjoerdpijpers$ javac OthelloAI16.java
(base) Sjoerd:Othello sjoerdpijpers$ java Othello DumAI OthelloAI16 8
Time AI_16:   0.00 s
Time AI_16:   0.05 s
Time AI_16:   0.11 s
Time AI_16:   0.01 s
Time AI_16:   0.53 s
Time AI_16:   0.93 s
Time AI_16:   1.66 s
Time AI_16:   1.52 s
Time AI_16:   2.66 s
Time AI_16:   5.33 s
Time AI_16:   6.53 s
Time AI_16:   2.05 s
Time AI_16:   1.19 s
Time AI_16:   6.36 s
Time AI_16:  10.47 s
Time AI_16:   0.48 s
Time AI_16:   0.21 s
Time AI_16:   0.07 s
Time AI_16:   0.11 s
Time AI_16:   0.14 s
Time AI_16:   0.12 s
Time AI_16:   0.01 s
Time AI_16:   0.07 s
Time AI_16:   0.08 s
Time AI_16:   0.06 s
Time AI_16:   0.02 s
Time AI_16:   0.00 s
Time AI_16:   0.00 s
Time AI_16:   0.00 s
Time AI_16:   0.00 s
```

Our OthelloAI16 wins from DumAI on the 6x6 and 8x8 board, however it doesn't win on the 4x4 board.