Umeå University
Department of Computing Science

Artificial intelligence
Methods and applications

5DV181

Othello artificial intelligence

Thomas Ranvier

supervised by
Ola Ringdahl Juan Carlos Nieves Sanchez

November 30, 2018

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Contents

Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Chapter 1

How to run my code

I slightly modified the 'othellostart' script, it is now placed above the 'naive' repository that contains the naive code, the way to use it is the following:

```
./othellostart .script_1 .script_2 t
```

Where '.script_1' and '.script_2' can correspond either to './naive/othello' or './java/othello' depending on the game disposition that you chose. The variable 't' corresponds to the time limit that you want to set.

So for example if you want to see a fame between the naive as white and my implementation as black for a limit time of 2 seconds this is the command to enter:

```
./othellostart ./naive/othello ./java/othello 2
```

Chapter 2

My implementation

2.1 Alpha Beta implementation

I implemented the Alpha Beta algorithm following the wikipedia pseudocode. My code can be found in the AlphaBeta.java file that contains the class which is used to determine the best move to do.

Here is the pseudocode of my implementation adapted to the Othello game.

Algorithm 1 Alpha Beta

```
1: procedure alphaBeta(initial_position, depth, alpha, beta)
2:     valid_moves ← initial_position.getValidMoves()
3:     if depth == 0 ∨ valid_moves.size() == 0 then
4:         return a move with value set to the evaluation
5:     end if
6:     maximizing ← initial_position.getPlayer() == this.player
7:     best_move ← a default move
8:     if maximizing == true then
9:         value ← −∞
10:        for all move from valid_moves do
11:            new_pos ← a clone of initial_position
12:            new_pos.makeMove(move)
13:            res_move ← alphaBeta(new_pos, depth − 1, alpha, beta)
14:            if res_move.value > value then
15:                value ← res_move
16:                move.value ← value
17:                best_move ← move
18:            end if
19:            if value ≥ beta then
20:                move.value ← value
21:                return move
22:            end if
23:            alpha ← max(value, alpha)
24:        end for
25:    else
26:        value ← ∞
27:        for all move from valid_moves do
28:            new_pos ← a clone of initial_position
29:            new_pos.makeMove(move)
30:            res_move ← alphaBeta(new_pos, depth − 1, alpha, beta)
31:            if res_move.value < value then
32:                value ← res_move
33:                move.value ← value
34:                best_move ← move
35:            end if
36:            if value ≤ alpha then
37:                move.value ← value
38:                return move
39:            end if
40:            beta ← min(value, beta)
41:        end for
42:    end if
43: end procedure
```

The recursive alphaBeta function returns an Othello move.

When the maximal depth is reached or if no other moves can be performed the function returns a move with a value set to the evaluation value. The evaluation value is determined by the evaluator which analyse the board of the game and return a value that is high if the evaluation for the actual player is 'good' and lower otherise. The evaluator uses a heuristic function that will be described later in this report.

The recursive function calls itself back and since when the 'makeMove' function is called the player is changed the function can automatically know if it must maximize or minimize its selection. Indeed when the function is selected a move for its own player the higher the evaluation score the better, this is why the function maximizes its selection in that case. On the other hand when it is selecting a move for the opponent if the evaluation score is low it means that it is a good move for the opponent so the function minimizes its selection in that case.

2.2 Iterative deepening

To handle the time limit the solution is to use iterative deepening for the alplaBeta function. There is an internal timer in the AlphaBeta class, this timer is checked at each iteration through the valid moves in the alphaBeta function and as soon as the time is reached the function throws an exception.

The alphaBeta function is initially called with a depth of 5 (since it is always way under 1 second) and then the depth is incremented by 1 and the function is called again with the new depth. That process repeats itself until it catches an exception, then the whole process stops and the move saved from the previous depth is returned as best move.

Here is the pseudocode of the bestMove function which implements the iterative deepening described above and also directly returns a pass move if the game is over.

Algorithm 2 bestMove

```
1: procedure BESTMOVE(initial_position)
2:     if initial_position.getValidMoves().size() == 0 then
3:         return a pass move
4:     end if
5:     this.timer ← System.nanoTime()
6:     depth ← 5
7:     best_move ← a default move
8:     while depth < 20 ∧ (System.nanoTime() − this.timer)/1000000000 <
   this.time_limit do
9:         try {
10:        best_move ← this.alphaBeta(initial_position, depth, −∞, ∞)
11:        depth ← depth + 1
12:        } catch {}
13:    end while
14: end procedure
```

2.3 Used heuristics

The heuristics function is what will evaluate a board in order to determine if doing a given move is a good idea or not. It is the part that differenciates the artificial intelligences from one another, indeed a well balanced and adapted heuristics function will perform way better than a naive heuristics function.

2.4 The makeMove function

I choose to realise the clone of the current position using another function which is directly called from the alphaBeta function, it is then not included in the makeMove function.

The first thing done in the function is to check if the asked move is in the grid boundaries and if it is not a pass move. Then it sets the cell selected by the move to the owned player. After that for all eight directions it calls the function checkDirection which returns true if the enemy cells on that direction are being captured. If it returns true the function fillDirection is called, this will capture the enemy cells by replacing them by the owned player. Once the process is done the player is changed for its opponent.

Here is the pseudocode of that function.

Algorithm 3 makeMove

```
1: procedure MAKEMOVE(x, y)
2:     if x and y are in boundaries then
3:         this.grid[x][y] ← this.player
4:         for dir_x ← −1; dir_x ≤ 1; dir_x + + do
5:             for dir_y ← −1; dir_y ≤ 1; dir_y + + do
6:                 if dir_x ≠ 0 ∨ dir_y ≠ 0 then
7:                     if this.checkDirection(x, y, dir_x, dir_y) then
8:                         this.fillDirection(x, y, dir_x, dir_y)
9:                     end if
10:                end if
11:            end for
12:        end for
13:        this.changePlayer()
14:    end if
15: end procedure
```

Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

List of Algorithms

List of Figures