Assignment 1
EDA132 Applied Artificial Intelligence

Fredrik Paulsson
dat11fp1@student.lu.se

Shan Senanayake
dat11sse@student.lu.se

February 12, 2015

1 Introduction

In this assignment we had the task to construct a program with a playable Othello game and an AI bot as the opponent. The game is played with ASCII art over the console. The program is written in Java and consists of three classes which are OthelloGame, OthelloAI and OthelloMain. These classes will be further explained later in this report.

2 OthelloGame

This class represents the Othello game itself. It manages a board with all the game pieces, players and the current turn with all valid moves. This class has methods to make moves and switch turns. In order to make the AI being able to simulate several turns of a game, this class includes a method which creates a new instance of this class with the same game state as the current game. This enables the AI to copy a game and simulate that game for several rounds in order to deduce the best move to make in the current round.

The class consists of two constructors, one of which is private, seven public methods and a couple of private methods. In addition to this it also contains fields and static variables that are used throughout the class to have a more dynamic use of variables.

2.1 Code

We will describe some interesting parts of the code in OthelloGame below.

2.1.1 Fields

`public static final char L` This character represents the 'Light' player outside of the class and board.

`public static final char D` This character represents the 'Dark' player outside of the class and board.

`private static final char E` This character represents an empty space on the board outside of the class and board.

`private static final char V` This character represents a valid move on the board outside of the class and board.

`private static final int EMPTY` This int represents an empty space on the board in the board matrix, this variable has the value 0 to make the heuristic function easier to compute.

`private static final int LIGHT` This int represents the 'Light' player on the board in the board matrix, this variable has the value 1 to make the heuristic function easier to compute.

`private static final int DARK` This int represents the 'Dark' player on the board in the board matrix, this variable has the value -1 to make the heuristic function easier to compute.

`private int[][] board` Represents the board of the game.

`private int player` Represents which player has the current turn.

`private int HashMap<String, LinkedList<Integer[]>> validMoves` Represents the valid moves which can be made by the current player, and which tiles will be flipped when that move is made.

2.1.2 Constructors

OthelloGame has two constructors, one public and one private. The public constructor is used to initiate the game. The private constructor is used to create a certain game in a certain state. This is used to make a copy of an already existing game.

2.1.3 Methods

This class has a lot of methods, both private and public, to cut down the complexity in this report only the most important parts revolving the AI are included in this report. The other methods handle print outs, making moves and conversions (between print format and such) as well as getters and setters that are needed for the AI and starting the program.

`private void findValidMoves()` This method finds all the valid moves the current player can make, as well as finds all the tiles that will be flipped and sets this as the `validMoves` map.

`private LinkedList<Integer[]> checkDirection(int x, int y, int incX, int incY)` The method `findValidMoves()` uses this method to check in a certain direction which tiles will be flipped, if a tile of the current player is picked. The direction is indicated by the `incX` and `incY` parameters.

`public int sumScore()` This method is the heuristic function currently being used, it sums the integers that make up the `board` variable, and returns the value gained.

3 OthelloAI

OthelloAI is the class that implements the AI part of the game. It interacts with the OthelloGame to be able to present a move that is as good as possible. This works by supplying a maximum processing time for the AI to respond with a move.

3.1 Algorithms

The algorithm used to determine the best move is the Minimax algorithm with Alpha-Beta pruning implemented. The Minimax algorithm is an algorithm which considers all moves for every turn that remains before the game is finished. It looks at all moves possible in the current turn of the game and for all those moves the next possible moves and so on. This means that all possible moves are tried out and finally one move is selected as the best. In order to select this move the algorithm always assume that the opponent makes the worst move from the AI point of view. Therefore the algorithm assumes the opponent makes the move which has the minimum of value of the heuristic function and then the algorithm selects the move that maximizes the heuristic function.

As can be realized it is extremely demanding to consider all possible moves for all possible turns in the game. To mitigate this we have done two optimizations. The first one is Alpha-Beta pruning which means that we do not consider moves which cannot be better than moves that has already been considered. However even with Alpha-Beta pruning the algorithm might take quite some time to compute the best possible move. To make the algorithm run in a reasonable amount of time we have given the AI a time limit for which it can run the algorithm. If the time limit is reached then it return the best possible move from that standpoint. Since the Minimax algorithm runs a depth first search (DFS), the resulting time limit might mean that the algorithm is still stuck in one branch and has not yet evaluated the next branches. To fix this we have done an iterative DFS. We run the Minimax algorithm one layer at a time, and between each layer it checks if the time limit has been reached, if not try to go one layer deeper. This will make a valid best move for the given time limit.

The Minimax algorithm can be visualized as a decision tree. Such a tree is diplayed in figure 1. Level 1 holds the moves that are possible for the algorithm to decide between and level 0 is the move that is selected. The values displayed in the nodes are the heuristic function's values. The algorithm has to select the move which has the maximum value in odd levels of the tree and it selects the moves with the minimum values in even levels of the tree. Alpha-Beta pruning works in the following manner. Example based on figure 1 If the right hand side of level 1 has been calculated the left hand side does not need to be fully calculated because when calculating the values of the children to the left hand side and the value -10 has been calculated no more work has to be done. This is because the values are minimized on level 2 in the tree and therefore it is guaranteed that the result of the left hand side on level 1 is at the greatest -10, but we already had a better value, -7, for the right hand side. Because on level 1 we maximize the values. Using this technique makes it possible to remove several nodes from from consideration and therefore remove costly operations.

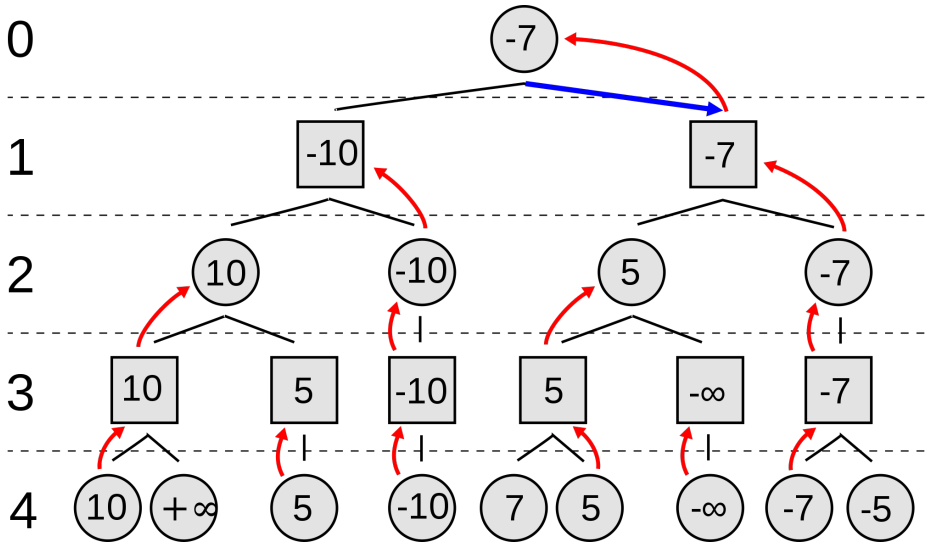

Figure 1: Minimax decision tree. Source: http://en.wikipedia.org/wiki/File:Minimax.svg

3.2 Heuristics

As described above a position of the game board has the value of 1, -1 or 0 depending on if the light or dark player owns the position or if the position is empty. We have decided to let the AI play as the player whose game piece is represented by the value 1. The opponent is thus represented as the value -1. Because of this we chose the heuristic function to be the sum of all the positions on the board. This means that the AI wishes to have as high a value as possible of this function because this means that it is the leading player.

Chosing the heuristic function like this is very natural for the Minimax algorithm. The AI wants to maximize it's lead and therefore it maximizes the heuristic function. It also assumes that the opponent selects the move which maximizes the opponents lead and thus minimizing the heuristic function.

The herustic function we have selected also has one big advantage and that is that it can be calculated exactly even though the game has not yet been finished. Therefore it is not essential for the Minimax algorithm to reach the terminal nodes before having values on the different moves. Of, course only considered moves will contribute to the heuristic function this way.

3.3 Code

Here the code of OthelloAI will be explained a bit more in detail.

3.3.1 Fields

`private OthelloGame othelloGame` This field represents the current game being played on, this is used by the AI to deduce the next move it should make.

`private long timeLimit` This field is the timeLimit that the AI has to run its algorithm on, if the timelimit is exceeded it should terminate the algorithm and return the currrently best move available.

`private long startTime` This field represents when the algorithm started running.

`private boolean isDone` This field checks when we are done running the algorithm and should return the best move, ergo when a leaf has been met.

3.3.2 Methods

`public String deduceMove()` This method is the only public method in the AI and is called for deducing the next move the AI should make. It initiates the Minimax algorithm by use of the `OthelloGame` it copies the board and and runs the DFS one layer deep at a time for which it stores the currently best move, and then if the time limit has not been exceeded runs the DFS one layer deeper until either time limit is exceeded or the algorithm has reached a leaf node.

`private int recursiveMax(OthelloGame game, int layer, int prevMin)` This method represents the 'Max' part of the Minimax algorithm, it recursivly calls the `recursiveMin` method until it either has reach the layer indicated by the `layer` parameter or a leaf node. It might call `recursiveMax` instead if the opponent was not able to make any moves in this round.

`private int recursiveMin(OthelloGame game, int layer, int prevMax)` This method represents the 'Mix' part of the Minimax algorithm, it recursivly calls the `recursiveMax` method until it either has reach the layer indicated by the `layer` parameter or a leaf node. It might call `recursiveMin` instead if the AI was not able to make any moves in this round.

4 OthelloMain

This class creates and initalizes the OthelloGame and OthelloAI classes. It also holds the main loop that lets the user and AI input moves to the initialized game for as long as there are valid moves left to make in the game.

5 Running Instructions

The three .java files containing the classes described above are located in a package called othello placed on /h/d9/v/dat11fp1/TAI/othello on the student computer system. It is simply to compile the java files in the package and then run OthelloMain from outside that package. For example if the current directory is /h/d9/v/dat11fp1/TAI the following command will compile the files: `javac othello/*.java`. After compiling the program can be run with the following command: `java othello.OthelloMain`.