Reversi Game - $\alpha - \beta$ adversarial search

Pablo Correa Gmez 960924T154
Jaime Saura Bastida 940718T313

February 8, 2018

Introduction

As a solution for the first assignment of the course, we have decided to implement a MinMax algorithm with $\alpha - \beta$ prunning included. The heuristic calculation is based on the amount of pieces that each player has at the end of the searching tree.

Code highlights

Code Structure

The Reversi game is implemented in Java and has three different modules:

- Game: Where the game play, initialization and conclusion can be found
- AI: Where the class implementing the algorithm can be found.
- Utilities: Where the interface for the human player and for the board and rules are placed.

Important methods

The top-4 methods of the implementation are:

- **ArrayList<Coordinates>** Board.possibleMoves(**int** player): First finds each one of the chips of the corresponding player and then looks around for chips of the adversary that has blank spaces on the other side. It is needed both to check that the human player is doing a correct move and to inform the algorithm which places are available to place a chip.
- **void** Board.move(**Coordinates** coords, **int** player): Given the coordinates and the player that is moving, places the chip on the appropiate coordinates and looks for the adversary chips that need to bo flipped.
- **void** AI.move(**Board** board): Prints some info about the AI chosen movement and initializes the minMax algorithm, as the tree root has the main singularity that the heuristic for it is irrelevant; only the movement associated with the heuristic is really significant. However, the first call for the algorithm has the same structure as the algorithm itself.

- **int** AI.minMax(**Board** board, **int** depth, **int** ref_value): This is the recursive algorithm. It has some initialization where we find out if we are in a minimun or in a maximum and initializes the reference value to the needed extreme (as we introduced $\alpha - \beta$ prunning) that is needed for the algorithm. This is needed because before finding the first heuristics of each branch, we do not have a valid reference to compare with. Then we have two possibilities:
  - If we are in the maximum depth, it iterates through all possible moves, calculate the heuristics of each of them and returns the minimum.
  - If we are not in the maximum depth then, for each move, we need to call the algorithm again, get its value, check with α or β if continuing with the search is needed and, in case it is, add the value to our possibilities so we can return later the maximum or minimum of all of them.

Game and usage

The game is represented by printing a Matrix of numbers, each number representing either if that position is a Black or a White chip or if it is Empty. When the game begins, the human player is asked the option to choose the color and the time (in seconds) that the algorithm will have to do the calculations. Then, before asking for a move, the game prints all possible ones and will not continue executing until a correct one is introduced. The game is finished when none of the players is able to do any move.