LUNDS UNIVERSITET

Lunds Tekniska Högskola

Building an Artificial Intelligence for Reversi

EDAF70 | Applied Artificial Intelligence | Project 1

Rachit Agarwal + Jay Mangrulkar

Table of Contents

Introduction

This project allows users to play the classic board game *Reversi* (commonly known as *Othello*) against an artificial intelligence. Here is a succinct description of Reversi (from Wikipedia):

> *Reversi is a strategy board game for two players, played on an 8×8 uncheckered board. There are sixty-four identical game pieces called disks (often spelled "discs"), which are light on one side and dark on the other. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled. Reversi was most recently marketed by Mattel under the trademark Othello.*

Program

The program is built using a single Python 2.7 file (`main.py`); the AI is built using the minimax algorithm with alpha-beta pruning. The minimax algorithm is a decision tree rule used in many games and statistics that minimizes the possible loss of a worst-case scenario, and when dealing with the best-case scenario, it maximizes the gain. The heuristic used by the algorithm is the Reversi score (number of player's tiles on the board). Alpha-beta pruning was used to makes the minimax algorithm more efficient by decreasing the number of nodes that are evaluated in the search tree.

The game is represented using an 8x8 grid board that was built using arrays (Python lists). Every move is represented using a tuple of a row and column (both integers between 0 and 7), but to the user this appears as a single string with a column letter and row number (i.e. "c6"). When starting the program, the user can choose how long to have the computer wait and what color to play as (black with "X" tiles, or white with "O" tiles). Once the game begins, for every move (both the user's and computer's), a list of valid moves is presented. If there are no valid moves for either player, the player's turn is "passed" or skipped. Once there are no valid moves for either player or the game board is filled, the game result is presented (user wins, computer wins, or tie). The user can also quit at any time.

Methods

Below is a list of all methods in the program, with short descriptions and relevant notes.

General Game and User Interaction

- **gameWelcome():** Welcomes user to game, gives relevant info and basic instructions.
- **printBoard(board):** Given a current game board (8x8 list of lists), prints in an 8x8 grid
- **printScore(board):** Given a current game board, computes scores for both sides and prints them to console.
- **selectTime():** Allows user to select CPU time delay in seconds.
  - No response or time < 0 entered: Defaults to 1 second.
  - Time > 10 entered: Defaults to 10 seconds.
- **selectColor():** Allows user to select which color/side to take.
  - No response or bad values (not “black”, “b”, “white”, or “w” entered): Defaults to black.
- **selectDepth():** Allows user to select depth for minimax algorithm.
  - No response: Defaults to 5.
  - Depth < 1 entered: Defaults to 1.
  - Depth > 10 entered: Defaults to 10.
- **gameOver(board, userColor):** Called only if game is over, prints final result and scores to console.
- **main():** Main method, includes user and computer move handling.

Board Functionality

- **createEmptyBoard():** Creates the 8x8 board (list of 8 lists), each tile initialized with “ ” (single space)
- **setupBoard():** Initializes board with starting tiles
  - Adds an “X” (black) at position D4 and E5
  - Adds a “O” (white) at position E4 and D5
- **printBoard(board):** Prints board to console by adding a row of letters a-h at the top and a column of numbers 1-8 along the left side. Also adds lines between rows and columns, and displays “X”and “O” for black and white, respectively.

Score Functionality

- **getScores(board):** Returns tuple of scores (number of tiles of each side/color) - (black, white)
- **getPlayerScore(board, playerColor):** Returns score for a given player/color

- **getTotalScore(board)**: Returns total number of disks on board (sum of 2 players’ scores)
- **printScore(board)**: Computes and prints scores to the console

Moves Functionality

- **getFlippedTilesList(board, moveColor, row, col)**: Returns list of (row, column) coordinate tuples of flipped disks for given player and move.
- **getFlippedTilesListString(board, moveColor, row, col)**: Returns string of list of flipped disks, but with column letters and row numbers (ex. “c7”), for given player and move.
- **getNumFlippedTiles(board, moveColor, row, col)**: Returns number of flipped tiles for given player and move.
- **flipTilesAndReturnNewBoard(board, moveColor, row, col)**: Given board, player, and move, flips the appropriate disks and returns the resultant board.
- **isValidMove(board, moveColor, row, col)**: Returns boolean of whether or not a given player’s move is valid, based of if any disks are flipped as a result.
- **getAllValidMoves(board, moveColor)**: Returns list of all valid moves for the given player/color.

Miscellaneous Functionality

- **convertColor(val)**: Converts a color string (“black” or “white”) to 1 or -1, used for flipping disks/tiles.
- **convertTileTupleToString(tile)**: Converts a given coordinate tuple (ex. (3, 4)) to a string with letter and number (ex. “d5”).

AI Functionality

- **makeCpuMove(board, moveColor, depth, validMoves)**: Given board, CPU’s color, depth, and list of valid moves, computes best move using minimax algorithm and returns that move
- **minimax(board, moveColor, depth, maximizing, alpha, beta, validMoves)**: Given board, CPU’s color, max depth value, maximizing or minimizing (boolean), alpha and beta values, and list of valid moves, runs minimax algorithm (with alpha-beta pruning) to compute best move
    - Evaluation function/heuristic used for a particular move is simple, the score (number of tiles on the board) as a result of that move for the player; the algorithm calls *flipTilesAndReturnNewBoard()* on possible moves, then evaluates the score of resultant boards using *getPlayerScore()*

Usage & Instructions

In order to run the program, enter the terminal/command line and navigate to the **`/ra3160ag/edaf70-p1-reversi`** directory. Then run the Python file with **`python main.py`**.

The program will start, and all instructions, game board, information, etc. is printed to the console. In order to play, first select the CPU time, user color, and algorithm depth, then make moves by entering the concatenated column letter and row number (ex. "c7"). To quit at any time, type "quit." Once the game ends, the result and score are presented and the program exits automatically.

The game's source code can be accessed through GitHub.