ECE 469 Othello AI

Casey He

November 20, 2017

1 Introduction

This project is an Othello/Reversi game written in C++ that utilizes a min-max search with alpha-Beta pruning implemented through iterative-deepening.

2 Project Specifications

2.1 Play Options

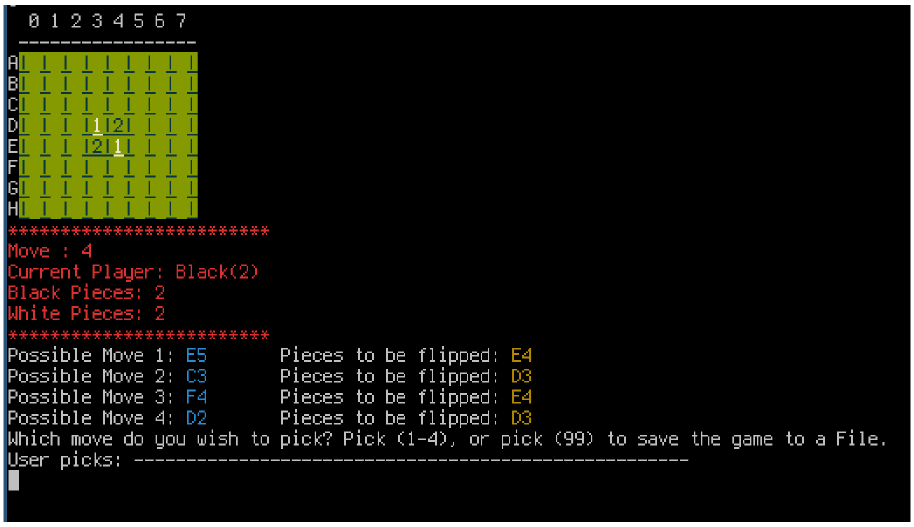
Users have options to play 2-player, against an A.I., or have an A.I. play itself. When playing against an AI, a user can pick which color piece to play (Black or White), and choose to set a time limit for the A.I. The default time limit is 5 seconds.

2.2 User Interface

The game of Othello is played on a green 8x8 board comprised of White Pieces and Black Pieces. The display uses xterm-256-color code support to display a Green Board with White "1" and Black "2". For every player move, The display outputs relevant game state information in a block of red text, such as what turn it is and how many pieces do each player have.

The display also provides a selection of all possible moves for the given game instance, along with all pieces flipped per given move. This provides the user with all the information necessary to make a good move.

A user, at any point, can also choose to save the current game state by inputting '99'. A user will then be prompted to provide a filename to save the file to, and can then load the game state at the beginning of the program to continue playing.



```
  0 1 2 3 4 5 6 7
 -----------------
A|_|_|_|_|_|_|_|_|
B|_|_|_|_|_|_|_|_|
C|_|_|_|_|_|_|_|_|
D|_|_|_|1|2|_|_|_|
E|_|_|_|2|1|_|_|_|
F|_|_|_|_|_|_|_|_|
G|_|_|_|_|_|_|_|_|
H|_|_|_|_|_|_|_|_|
*************************
Move : 4
Current Player: Black(2)
Black Pieces: 2
White Pieces: 2
*************************
Possible Move 1: E5       Pieces to be flipped: E4
Possible Move 2: C3       Pieces to be flipped: D3
Possible Move 3: F4       Pieces to be flipped: E4
Possible Move 4: D2       Pieces to be flipped: D3
Which move do you wish to pick? Pick (1-4), or pick (99) to save the game to a File.
User picks: -----------------------------------------------------
```

User Interface screen

2.3 Search Algorithm

The search algorithm is a straight-forward minimax search with alpha-beta pruning and iterative deepening. Near the end of the game, the AI will search for terminal states to help aid in a victory. The most important aspect in building a fast search algorithm is the way valid moves are calculated. This is because the limiting factor in how many nodes traversed by the search tree is how long it spends at each node. The algorithm starts by scanning a 2-D array for the pieces of the current player. When it reaches a piece, it then searches outward in 8 directions looking for viable moves. Possible pieces to be flipped are contained in a list. When a valid move is found, the coordinates of the move are used as a unique 'key' and paired with the list of flipped moves. This pair is then inserted into a hash table 'std::unordered_map'. Multiple lists of flipped pieces can be given for a single move– thus the lists are concatenated together. Once the the 2-D array is scanned, all possible moves are now stored in the hash table, along with all the pieces needed to be flipped. By using a hash table, we can ensure constant time insertion and lookup.

2.4 Heuristic

The Heuristic was created with efficiency in mind. The heuristic function evaluates the game board with a linear combination of various parameters, including:

- Parity Which Player has the Final Move
- Coin Ratio A ratio of Player Pieces vs. Opponent Pieces
- Terminal State Whether a positive terminal State can be reached
- Pivotal Squares The establishment of Strong Corners

2.5 Acknowledgements

- George Ho for discussion of Othello Search Strategies
- Brian Hong for playing against my A.I. to test its strength
- Professor Carl Sable for providing rigorous project specifications and a challenging project