Project 1 Report

Rob Owens

CSCI 440

rtowens@stcloudstate.edu

ABSTRACT

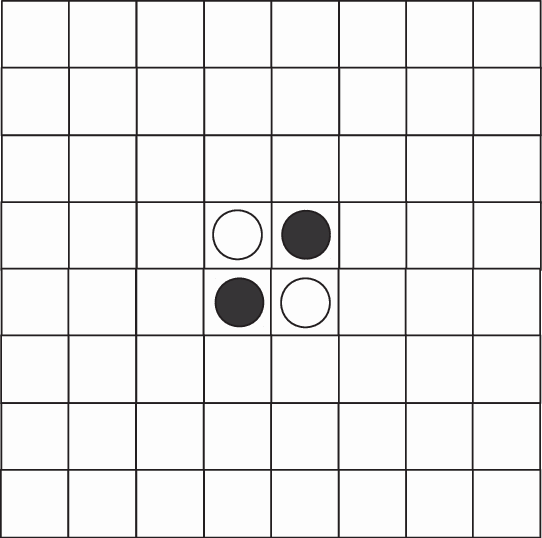
This program is an artificial intelligence that plays the game Othello. This program is not just simply a lookup table and generates a tree of possible moves and then searches the tree for the best move to make. The program does not sport many features outside of the basics but the emphasis in this instance is the AI. The AI performs well in practice, being undefeated in testing against humans and other tested state evaluation functions.

0 Introduction

This program is designed to play the game of Othello and implement an artificial intelligence (AI) to play against. The program is rather minimalistic but plays the game well and in testing beats most average human opponents. By implementing alpha beta pruning to the tree searching the AI can search farther down into the tree and predict at least 6 moves ahead, if not more.

1 The Game

The game of Othello is about placing pieces on a board in the simplest terms. On each player turn they will place a piece such that they “capture” or flip an opponent’s piece. This is done by having one of your pieces on either side of 1 or more of your opponent’s pieces in a straight or diagonal line. Thus, after each turn not only is one piece added but at least one piece has changed sides as well. This goes until a player cannot move at which point the player with the most pieces on the board wins. The starting position is shown below, and in every game the black player goes first.



2 Design Choices

The first major design decision was to go with a text interface instead of using a program like unity to make a make friendly interface. This allowed me to spend more time working on the artificial intelligence behind the program and improving the performance then fixing issues with the interface. Following that, another decision that saved time and resources was the decision to make the game a class that was self-contained so that the AI just had to interact with the board and the code was separated. This allowed me to develop and test the game itself before making the AI so that upon creating the AI, I knew all the issued were in the AI class and didn’t have to touch the game class. This proved to save lots of time and sped up development by quite a large margin. A final design decision was to make the AI a class so that during development when trying to fix a problem, there were less function calls and tracing required, thus making it easier to fix problems.

2.1 Features of the program

The program is fairly limited on features; however, the program will display the predicted score of the move it makes along with the move that is making. This not only allows for easier troubleshooting but provides a glimpse into how the program “thinks” it’s doing. Other than those features, the only other feature of the program is the AI itself which is covered in more depth in section 3.

3 Static Evaluation Function

The Static Evaluation Function (SEF) of the AI is a relatively simple one but is effective and does not take large amounts of processing power. If the game is over, it determines who is the winner and returns a corresponding value. If the game is not over, then each piece on the board is categorized into one of 3 categories: stable, flippable, and basic. The stable category is for pieces that cannot be flipped given any number of moves in the future. The way for searching for these pieces is for each piece we check each of the axis, if each axis contains one side that is purely all the same color pieces till the edge then that piece cannot be flipped as there is no legal move that would put this piece between 2 opponent’s pieces. The next is the opposite of these where the opponent could make a move this turn that would flip this piece. This is a similar search as the last one except we are check for an axis where there is a space on one side of the piece, opponents piece on the other side and only our pieces in the middle. If a piece does not fit either of these categories, then it is considered a basic piece and we don’t do any checks besides that it is a piece on the board.

As the pieces are being analyzed, weights corresponding to category are also applied to the pieces. Overall stable pieces are worth more points than basic pieces which are worth more points than flippable pieces. This makes the program prefer positions where it’s pieces can’t be flipped and where opponents have fewer moves or moves that do less. This leads to faster calculations on the program side as there are fewer moves to compute and/or the moves will do less. This also means that the program can also look deeper into the tree as if each state only has 2 or 3 possible moves, the tree will not grow as fast as scenarios where there are 5 or 6 possible moves.

3.1 Tree Searching

The tree of potential moves is searched using a implementation of alpha beta pruning. This reduces the number of leaves the program needs to compare and generate values for and combined with the SEF function which favors boards where the opponent has less moves, means the tree is rather trim at points. Thus, to take advantage of when the tree is not wide, if we are at a leaf where there are fewer than 3 potential moves it will keep searching till it opens or hits a dead end. Early on this does not provide much of a difference but later in the game where there are only 8-10 possible moves left in the whole game, the program can identify almost all game over states and play towards the game over states where the program wins. This overall allows the program to by default look 6 moves ahead but can sometimes reach depths of 8-10 moves ahead with minimal impacts on performance.

4 Using the program

Once the program is compiled the use of the program is relatively straight forward. Upon launch the program will ask which player should go first and then on each of your turns the current board will be displayed, and it will ask for your move. Simply input your row and column of the move, the game will make that move and then respond with a move of its own. This will repeat until the game is over at which point the winner will be declared and the program will exit.

5 Performance

The overall performance in terms of the game is good. I have yet to beat this program in its final form and this SEF and tree searching combination was tested against a couple different SEF and tree searching methods. Most notably it was compared to a basic SEF of just counting the pieces on the board and compared to a SEF based on number of potential moves for each player. Since Othello has no randomness in it, a simple game will suffice for the comparison and then switch which player is starting first and play again. This best of two will generally produce a winner, which means that the winner will always beat the losing SEF as the starting state is the same, so the same moves will be reached.

Another performance boost is in the game class itself, with numerous performance tweaks in the detection of legal moves and computing moves. These included checking each axis instead of each direction reducing the 8-direction check down to a 4-direction check. Another tweak was to detecting game over states, in where if either player has at least 33 stable pieces, meaning pieces that cannot be flipped, that board can be considered like a game over state and we don’t need to do any more checking on this board, no series of moves will change the outcome meaningfully. A final performance tweak that drastically improved how long it takes to computer a move is the implementation of alpha beta pruning. This allows for less states to be evaluated reducing the number of times boards need to be scanned which allows for a greater depth to be searched. Overall this allows the program’s AI to compute at least 6 moves ahead and sometimes more without a large impact on performance

6 Conclusion

In general, the program performs well and is reasonably fast to output its next move. While there is not a lot of complexity or fancy features the program’s, emphasis is on the AI and that is where the program’s best work is.Conference Name:ACM Woodstock conference

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00