Programming Assignment #2

Due May 4 by 5pm **Points** 50 **Available** after Apr 17 at 10:50am

CS 331 Assignment #2: Simplified Othello

Out: April 17, 2020

Due: May 4, 2020 at 17:00:00

Type: **Individual** assignment

Othello (also known as Reversi) is a classic two-player, turn-based game in which players try to flip as many pieces over to their color as possible. If you are not familiar with the rules for Othello / Reversi, please read the following **Wikipedia page** **(http://en.wikipedia.org/wiki/Reversi)** . You may also play a game on the following **webpage** **(http://www.gamesforthebrain.com/game/reversi/)** . In this assignment you will convert a simplified 4x4 two-player game of Othello into a one-player version in which a human player gets to play against a very smart computer opponent. You will be implementing the computer opponent using the Minimax algorithm discussed in class.

Requirements

You will calculate the next move for the computer player using the Minimax algorithm. Since we will be using ASCII art to display the board, we will use the symbols X (for the dark player who goes first) and O (for the light player who goes second). We will operate under the assumption that the player going first is the maximizing player and the player going second is the minimizing player.

The command line allows you to select whether a player is a human player or a computer player. We will only be grading your assignment with the human player moving first and the Minimax player moving second. Note that the 4x4 game of Othello is asymmetric; the player moving second has a serious advantage over the player moving first. In this assignment, the computer player, when going second, should always either win or tie.

The specific things you need to implement for this assignment are:

1. **Utility function:** The 4x4 version of Othello is small enough that we can generate the entire game tree while doing the Minimax search. As a result, you do not need to create an evaluation function for non-terminal nodes. You will, however, need to create a utility function that determines the

"goodness" of a terminal state. You will need to create this utility function on your own. When creating the utility function, remember that the player that moves first is assumed to be the maximizing player.

2. **Successor function:** You will also need to create a successor function. This function takes the current state of the game and generates all the successors that can be reached within one move of the current state.
3. **Minimax function:** To implement minimax, you will need to refer to slide 22 of the AdversarialSearch1 lecture. You will need to implement the Minimax-Decision, Max-Value and Min-Value functions on that slide. If you use the C++ code, you will need to fill in the get_move function of the MinimaxPlayer class.

In addition to the functionality described above, you may need to implement some other code to do things like bookkeeping.

Code

You will be provided with a partial C++ implementation of an ASCII version of the game. This code is for 2 human players and it implements the rules of the game. The minimax player is partially implemented and does not work in its current form. You will need to fill the MinimaxPlayer class in.

C++ Code (If you use C++)

The code can be downloaded as a **zipped archive**.

I encourage you to read through the code. The header files have detailed comments to explain the member functions and member variables. To help you navigate the code, here is a brief description of the classes:

1. **GameDriver:** This class runs the game by managing the turn-taking and the game display. It contains the main function.
2. **Player:** This is an abstract base class for a player in the game. Note that each player has an associated symbol, which is the symbol used for their pieces on the board (eg 'X' or 'O').
3. **HumanPlayer:** This class accepts console input to specify the moves
4. **MinimaxPlayer:** This class implements a Minimax player agent. You will need to fill in the get_move() function
5. **Board:** This class is basically a wrapper for a 2D array. It represents a generic board for a game.
6. **OthelloBoard:** The OthelloBoard class is a subclass of the Board class that is specialized for Othello. Functions in this class that could be handy include:
    1. **int count_score(char symbol)**: counts the number of pieces for a player (indicated by that player's symbol)
    2. **bool has_legal_moves_remaining(char symbol)**: if false, the game ends.

3. **void play_move(int col, int row, char symbol)**: puts a piece on the board for a given player (specified by the symbol arg) and flips the in-between pieces.
4. **char get_p1_symbol()**: gets the symbol for player 1
5. **char get_p2_symbol()**: gets the symbol for player 2

Python Code (If you use Python)

A former student translated the C++ code above to Python, and your TAs have approved its use. The code can be downloaded as a **zipped archive**.

Other code (if you reimplement from scratch)

You may choose not to use the provided code and re-implement everything on your own using C, Java, or a programming language approved by the TAs. If you write everything from scratch, you must have the following:

1. Your executable must take two command line arguments like the C++ program. These two arguments are the types of the players (human or Minimax). An example of a command line is: *othello human minimax*.
2. You must provide a user interface, which can be either ASCII or a GUI. The player that moves first is assumed to be the maximizing player. Your user interface must display the board and display the score for each player.

Hand in

All of your source. Zip everything up with a zip program. To hand in your assignment, go to the TEACH electronic handin site: **https://teach.engr.oregonstate.edu/ (https://teach.engr.oregonstate.edu/)**.

1. Login to the TEACH system
2. Click on the "Submit Assignment" link on the left hand side
3. Select ProgAssn2 from the dropdown menu, hit submit query
4. Enter the path of your zip file. Hit Submit Query to hand everything in.

Approximate Grading Criteria

- Utility function (5 points)
- Successor function (15 points)
- Minimax function (25 pts)
- bookkeeping code (5 pts)

Correctness of your code will be determined primarily by the markers interacting with the game. If the computer player cannot be beaten, you will get full credit.

Othello		
Criteria	**Ratings**	**Pts**
Utility Function		5.0 pts
Successor Function		15.0 pts
Minimax Function		25.0 pts
Bookkeeping Code		5.0 pts
		Total Points: 50.0