****

**Sheharyar Qureshi**

**Project Title: Game of Othello using Alpha Beta Pruning**

# Introduction:

Othello is a modern iteration of the game ‘Reversi’. This is a strategy board game for two players, one plays black and the other white. Each player gets 32 discs and Black always moves first. In my game the player is Black and the AI plays white. The game takes turn between black and white until one player

• Cannot make a valid move to outflank the opponent

• Both players have no valid moves remaining

• When a player has no valid moves, the turn is passed to the opponent

• A player cannot voluntarily forfeit his/her turn.

• When both players cannot make a valid move, the game ends. Player with most color tiles wins the game.

The game will appear on the console in the form of a 9x9 matrix with the first row and column numbering the rows and columns of our 8x8 game. A player will take turn by providing the row and the column index of a valid move otherwise the player will be required to repeat his turn to make sure a valid move is input for the game to move forward. Black will be represented by P and white by C on the grid. The game will appear on the console. I have attached a picture of the game in figure 1 below.

After every valid move, next 3 possible moves will be shown on the console. The AI will make its move dependent on the next 3 moves. After AI has taken its turn, the board will again be updated to continue the game. This will go on until someone wins.

# Rules and Valid Moves:

1. A move is made by entering the row and column where you wish to place your disc on the board in a position that "out-flanks" one or more of the opponent's discs.
2. A disc or row or column of discs is outflanked when it is surrounded at the ends by discs of the opposite color.
3. A disc may outflank any number of discs in one or more rows in any direction (horizontal, vertical, diagonal).
4. All the discs which are outflanked will be flipped, even if it is to the player's advantage not to flip them.
5. Discs may only be outflanked as a direct result of a move and must fall in the direct line of the disc being played.
6. If you can't outflank and flip at least one opposing disc, you must pass your turn. However, if a move is available to you, you can't forfeit your turn.
7. Once a disc has been placed on a square, it can never be moved to another square later in the game.
8. There are total of 64 places on the board. There are 4 tiles already placed at the beginning leaving 60 turns to be taken between the players.

# Logic:

I have used alpha-beta pruning to predict the next 3 moves which include 2 moves of the AI and 1 move of the player. I have used object-oriented programming to implement two classes. Moves class will contain the coordinates and count. Othello class contains all the functions for the AI and the player turns, the movement, etc. – essentially everything. I have had to implement the following functions in Othello:

1. bool checkBounds(int i):

This function checks if the input coordinates are well within the bounds of the board.

1. void printBoard():

This function sets the main board of the game on the console.

1. void EndGame():

This function renders the result of the game.

1. int score(vector < vector<int> > &board):

This function counts the scores and returns the difference between the amount of Player and AI tiles.

1. void convertUpRight()

This function will convert the tiles of the Player to the tiles of CPU or vice versa.

void convertUpLeft()

void convertDownRight()

void convertDownLeft()

void convertUp()

void convertDown()

void convertLeft()

void convertRight()

All these functions perform the same except for their direction.

1. int countUpRight()

This function will count how many tiles can be converted from Player to CPU or vice versa.

int countUpLeft()

int countDownRight()

int countDownLeft()

int countUp()

int countDown()

int countLeft()

int countRight()

1. int checkMove()

This function counts all the possible tiles from all directions using the family of all the count functions mentioned before.

1. void printAllPossibleMoves()

This function displays all the possible moves that can be made considering all directions.

1. int playMove()

This function is responsible to play the move. It will check all the directions and return the possible converts.

1. int AlphaBeta()

This is the function where alpha-beta pruning is implemented. It includes two sections, if it is the player’s turn it will go to the int\_min to minimize the player score. If it is the AI’s turn it will go to int\_max to maximize the AI’s score. If it reaches a point where it can not maximize or minimize any more it will prune.

1. int computerMove()

This function will perform similarly to the previous function except for the AI.

1. void playGame()

This function will create the board, take the player inputs and check if they are valid, call for the computer move and switch the player turns.

# Picture:

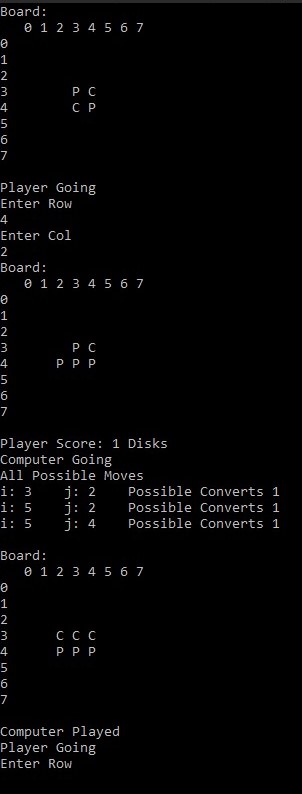


Figure 1: Othello