COMP10050 Assignment 3: Othello/Reversi Full Game (Documentation)

	Name	*Gitlab Username*
Player 1	Bryan	@letsdie10
Player 2	Xuan	@grasscait

Table of Contents

1. Questions from PDF

How you decided to implement the different methods used to implement the game logic?

Initialization

Before we even decided to implement any methods, we knew that our struct design might be flawed to begin with. The code handed in for assignment 2 was sufficient to for that assignment's specification, but not for assignment 3, therefore our first plan was to make sure our struct design was correct. We achieved this by referencing the struct design given in moodle.

Manage Turns

We knew we needed a turn managing function, but this will only be useful once we have the core codes ready (i.e. functions that handle the moves), therefore, this function was left to be implemented near the completion of the manage moves function. After the completion of manage moves function and making sure that it works, we implemented manage turns, which is as simple as creating a while loop where the end game condition is not met (i.e., there are still spaces on the board left to place disks).

Manage Moves

This is considered by us, to be the core function that runs the game. Therefore we tackle creating this part of the code head-on first. In this function, we will need a few things.

1. *Get coordinates of possible placement of moves by the current player.* We will first have to find current player's own disk. Let's say current player's disk's colour is black. We will first have to find black disks. This can be done by searching through the board, looking for black disks. If it is a black disk, we will have to check the perimeter for any opponent's disk (i.e. white disks). If there are white disks, we search further towards that direction relative to the starting black disk for more white disks. We stop searching when we reach either a black disk or a NONE (i.e., no disks). If we reached a black disk, we ignore that direction because it means we cannot place any disks at that direction. However, if we reached a NONE, it means we can place a white disk at that spot, and if the player chooses to place a white disk at that spot, all black disks between the starting white disk till the white disk where the player places the new white disk, will be converted to white disks. All possible moves are inserted and stored in a linked list.

2. *Display coordinates of possible placement of moves to the user.* This is achieved simply by going through the linked list and printing those coordinates to the console. An index variable i, is used to keep track of the number of possible moves. This is then stored in max.

3. *Prompt user for choice of move to make.* Scanf is used to acquire user's choice. A do while loop is used to check if the user entered a number outside of the possible range (i.e., not within 1 to max).

4. *Acquire chosen coordinate from user.* We first check if the linked list is empty, a secondary precaution to prevent segmentation fault (i.e., accessing memory that we shouldn't). Following that, if the linked list is not empty, we go through the linked list until we reach the coordinates the player chose. This is done by using an index variable i to keep track of the number of linked list nodes we went through. Once i equates to user's index choice, means we reached the correct node that houses the coordinates the player chose. Then we dequeue all the nodes in the linked list, to prepare it for the next turn.

5. *Place disk and update board with changes caused by placing that disk.* First use an if statement to check if that spot is occupied by any other type of disk. If not, proceed to place the disk and update the board. Let's say the current player's disk is white. To update the board, we check the perimeter for any black disks. If there are, we search further towards that direction relative to the spot we just placed our white disk. We keep searching until we reach either a white disk or NONE. If it is a none, it means we can't convert any black disks in that direction to white disks (because we need to be searching for the possible white disks that allowed the placement of the possible white disks). However, if it is a white disk, we can convert all the black disks between the white disk we just placed and this white disk.

Why use linked list instead of a simple 2D integer array to store number of possible moves

Initially, a simple 2D integer array was used to store the number of possible moves that can be made each turn by the respective players, however, we were recommended to use either dynamic memory allocation (i.e. calloc, malloc and free) or linked list. Considering the fact that although linked list can be quite hard to implement as it requires clear understanding of pointers, if done right, provides significant flexibility memory-wise and in terms of being able to insert data anywhere within the linked list, as well as flexibility memory-wise and in deleting data from anywhere within the linked list. This is demonstrated from the fact that the coordinates of the possible moves are inserted into the linked list in ascending order of the rows first, then columns.

Manage Ending

When manage turns function terminates (i.e., no board space left to place disks), manage ending function will be executed. Final results will be printed to the board, a refurbished printBoard function is used to print the board to the file, and the players' name, score and winner of the game is printed to the txt file as specified in assignment specification.

How did you divide the work (e.g., who did what)?

tl;dr,

Who	*Did What?*
Bryan	Rewrote and Prepared previous code, i.e., modified struct design.
Bryan + Xuan	Discussed game logic structure, i.e. manageTurns + manageEnding.
Bryan	Wrote game logic structure.
Xuan	Refactored chunks of code.
Xuan	Created manageEnding.
Xuan	Wrote code specifications, modularized code into modules.
Bryan	Further refactored and commented code for ease of future reference.
Bryan	Drafted readme.pdf.
Xuan	Finalized readme.pdf.

2. Implementation Details

Programming Language – C

Program Information – Othello/Reversi Game

PART 1 – Start condition. (Wherever Assignment 2 left off, with some changes)
PART 2 – Managing turns.
PART 3 – Printing the final result.

Function Information

***NOTE*:** More specific information regarding the respective functions are located in their respective files above their function definitions.

Function Information	*Function Location*
main() main function that is used to run the application.	main.c
initializePlayers() Function asks user for input and inputs players' name into the struct variable.	initialize.c
removeNewline() Function removes newline at the end of a string in a 1D character array.	initialize.c
initializeBoard() Function initializes disks positions on board to start conditions.	initialize.c
printBoard() Function outputs the board that represents the current turn.	initialize.c
manageTurns() Function that manages each turn by keeping track of number of turns and spaces left on board.	turns.c
manageMoves() Function that manages each move by calculating the number of possible moves, prints them, promts the user to choose one, then update the board relative to the choice made by the user.	turns.c
getCoords() Function gets coordinates by finding the possible moves.	turns.c
printCoords() Function prints the possible moves/coordinates to the console.	turns.c
askChooseCoord() Function prompts the user to choose a move to make.	turns.c
getChosenCoord() Function that check own disks in order to replace opponent's disks after each turn and updates each player's number of disks.	turns.c

isEmpty() turns.c
Function checks if the linked list is empty.

dequeue() turns.c
Function removes the node from the linked list as well as free the memory allocated to it.

placeDisks() turns.c
Function places user disks according to their choice and make changes to opponent's disks.

findPossibleMoves() findPossibleMoves.c
Function checks for opponent disks to determine possible positions current player can move to.

insert() findPossibleMoves.c
Function inserts the data i.e., row and column, into the linked list.

isInList() findPossibleMoves.c
Function loops through the linked list to check if the coordinates already exists within the linked list.

findMovesAtNorthWest()
findMovesAtNorth()
findMovesAtNorthEast()
findMovesAtWest()
findMovesAtEast()
findMovesAtSouthWest()
findMovesAtSouth()
findMovesAtSouthEast() findPossibleMoves.c
Functions above aid findPossibleMoves to look for possible moves in their respective cardinal directions.

updateDisksAndScore updateDisksAndScore.c
Function updates both player's disks based on current user's choice of move.

updateAtNorthWest()
updateAtNorth()
updateAtNorthEast()
updateAtWest()
updateAtEast()
updateAtSouthWest()
updateAtSouth()
updateAtSouthEast() updateDisksAndScore.c
Functions below aid updateDisksAndScore to look at their respective cardinal directions.

manageEnding() end.c
Function determines win/draw/lose conditions and prints out end game results.

saveToFile() end.c
Function outputs end game result to a txt file.

printBoardToFile() end.c

Function is a modified version of the printBoard function that is used to print data to files

Function Branching

Code Structure / Function Branching	*Function File*
main()	main.c
initializePlayers()	initialize.c
removeNewline()	initialize.c
initializePlayers()	initialize.c
main()	main.c
initializeBoard()	initialize.c
main()	main.c
printBoard()	initialize.c
main()	main.c
manageTurns()	turns.c
manageMoves()	turns.c
getCoords()	turns.c
findPossibleMoves()	findPossibleMoves.c
findMovesAtNorthWest()	findPossibleMoves.c
insert()	findPossibleMoves.c
isInList()	findPossibleMoves.c
insert()	findPossibleMoves.c
findMovesAtNorthWest()	findPossibleMoves.c
findPossibleMoves()	findPossibleMoves.c
findMovesAtNorth()	findPossibleMoves.c
insert()	findPossibleMoves.c
isInList()	findPossibleMoves.c
insert()	findPossibleMoves.c
findPossibleMoves()	findPossibleMoves.c
findMovesAtNorthEast()	findPossibleMoves.c
insert()	findPossibleMoves.c
isInList()	findPossibleMoves.c
insert()	findPossibleMoves.c
findMovesAtNorthEast()	findPossibleMoves.c
findPossibleMoves()	findPossibleMoves.c
findMovesAtWest()	findPossibleMoves.c
insert()	findPossibleMoves.c
isInList()	findPossibleMoves.c
insert()	findPossibleMoves.c
findMovesAtWest()	findPossibleMoves.c
findPossibleMoves()	findPossibleMoves.c
findMovesAtEast()	findPossibleMoves.c
insert()	findPossibleMoves.c
isInList()	findPossibleMoves.c
insert()	findPossibleMoves.c
findMovesAtEast()	findPossibleMoves.c
findPossibleMoves()	findPossibleMoves.c
findMovesAtSouthWest()	findPossibleMoves.c
insert()	findPossibleMoves.c
isInList()	findPossibleMoves.c
insert()	findPossibleMoves.c
findMovesAtSouthWest()	findPossibleMoves.c
findPossibleMoves()	findPossibleMoves.c

```
                                findMovesAtSouth()                              findPossibleMoves.c
                                        insert()                                findPossibleMoves.c
                                                isInList()                      findPossibleMoves.c
                                        insert()                                findPossibleMoves.c
                                findMovesAtSouth()                              findPossibleMoves.c
                        findPossibleMoves()                                     findPossibleMoves.c
                                findMovesAtSouthEast()                          findPossibleMoves.c
                                        insert()                                findPossibleMoves.c
                                                isInList()                      findPossibleMoves.c
                                        insert()                                findPossibleMoves.c
                                findMovesAtSouthEast()                          findPossibleMoves.c
                        findPossibleMoves()                                     findPossibleMoves.c
                getCoords()                                                     turns.c
        manageMoves()                                                           turns.c
                printCoords()                                                   turns.c
        manageMoves()                                                           turns.c
                askChooseCoord()                                                turns.c
        manageMoves()                                                           turns.c
                getChosenCoord()                                                turns.c
                        isEmpty()                                               turns.c
                getChosenCoord()                                                turns.c
                        isEmpty()                                               turns.c
                getChosenCoord()                                                turns.c
                        dequeue()                                               turns.c
                getChosenCoord()                                                turns.c
        manageMoves()                                                           turns.c
                placeDisks()                                                    turns.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtNorthWest()                             updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtNorth                                   updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtNorthEast()                             updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtWest()                                  updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtEast()                                  updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtSouthWest()                             updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtSouth()                                 updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                                updateAtSouthEast()                             updateDisksAndScore.c
                        updateDisksAndScore()                                   updateDisksAndScore.c
                placeDisks()                                                    turns.c
        manageMoves()                                                           turns.c
manageTurns()                                                                   turns.c
        printBoard()                                                            initialize.c
manageTurns()                                                                   turns.c
main()                                                                          main.c
manageEnding()                                                                  end.c
        saveToFile()                                                            end.c
```

```
                printBoardToFile()                    end.c
            saveToFile()                              end.c
        manageEnding()                                end.c
main()                                                main.c
```