**Date: 02/12/2018**

**Student Nam: Ayodeji Shote**

# **Reversi Game**

## **Statement of requirements**

Statement: I was tasked to creating a program in java that not only imitates the game reversi, but also the board game. This program must take the inputs for this game, these inputs which are numerical data are then processed by the game as the placements of the game pieces achieved by inputs of the x and y coordinates of a game piece by the user. The outputs for the game is the visual representation of the boards changing, due to the position of the pieces placed down on the program console window every time there is an update to the board the program prints a new board .

## **User assumptions:**

The user is over 10 years of age.

The main form of input is by inputting the x position and the Y position.

The user can use a computer for the program.

### **Requirements**

**Functional Requirements:**

R1: The pieces shall have two options, the X piece or the O piece, one for each player.

R2: The X piece will always go first.

R3a: A piece will change its type when an opposing piece is placed beside it horizontally .

R3b: A piece will change its type when an opposing piece is placed beside it vertically .

R3c:A piece will change its type when an opposing piece is placed beside it diagonally.

R4: The game shall have a grid board with 8x8 dimensions.

R5: The game shall boot up into a start menu that displays the message welcome to reversi .

R6: The start menu shall have a start game option.

R7: The start menu may have a load game option.

R8: The start menu shall have an exit game option.

R9: A piece shall only be allowed to be placed in a square if it results in at least one opponents piece being flipped.

R10: The current game shall end when there are no more squares to play on or when one of the two players has no valid moves .

R11: The current game shall end when all of the pieces on the board are the same colour.

R12: The game shall have an auto save option before the piece X’s turn of the game.

R13: One player shall be a user while the other shall be another player.

R14: One player shall be a user while the other shall be played by the computer.

R15: The user should be able to select their names.

**Non-functional Requirements**

NF1: The game shall be able to run on Windows, Mac and Linux with no errors.

NF2: The game shall not have copyright material present in it .

NF3: The loading screen shall take no longer than 3 seconds when loading the game and its contents .

NF4: The loading of a saved game shall take no longer than 3 seconds its contents .

NF6: The program’s language should be english.

### 

### 

### 

### 

### **User Interface:**

Player 2 name

Number of O pieces:

Player 1 name

Number of X pieces:

1 2 3 4 5 6 7 8

1

2

3

4

5

6

7

8

Displays player names

Displays player Score

Yellow highlight used to show the current turn.

**Player VS Player :**

Welcome to Reversi

**Player VS AI:**

**Load Game:**

**Exit Game:**

**Title**

**The load game loads previous autosave**

### **Use Cases**

##### **Candidate Classes**

|  |  |  |
| --- | --- | --- |
| **Candidate Classes** | **Accept / Reject** | **Reason for rejection** |
| Othello | Reject | Name of the game played not an actual class needed for program |
| Reversi | Reject | Repeat of Othello, not actually a class needed |
| Board game | Reject | This is too simplified for what it actually represents for the program |
| Players | Accept | This holds the information about the players and handles the user input. |
| Grid | Accept | Holds the information about the game and plays the game. |
| Pieces | Accept | Holds the information about the individual pieces |
| Two Distinct Sides | Reject | Repeats of pieces |
| X Piece | Reject | Attribute of Pieces |
| O Piece | Reject | Attribute of Pieces |
| Game Board | Reject | Repeat of Grid |

##### 

##### **Class Descriptions including Responsibilities, Fields and Methods**

|  |  |  |
| --- | --- | --- |
|  | Class | Attribute/Method |
| 1. | Players | Player2X(M)  Player2Y(M)  Player1X(M)  Player1Y(M)  setNames(M)  yourname1(M)  yourname2(M) |
| 3. | AI (inheritance) | AI(input) |
| 4. | Grid | validMovesp1E(A)  pointCounterX(A)  p1validN(A)  intArray(M)  displayArray(M)  gameBoard(M)  Save Game (M)  Valid Move Counter(M)  Move Validator(M)  Number of valid moves(A)  Booleans for move directions (A)  Game(M)  Endgame(M)  Winner(M) |
| 7. | Start Screen | displayMenu(M)  Player VS Player (A)  Player VS AI(A)  Menu(M)  Load (A)  Quit (A)  fileReader(M) |

##### 

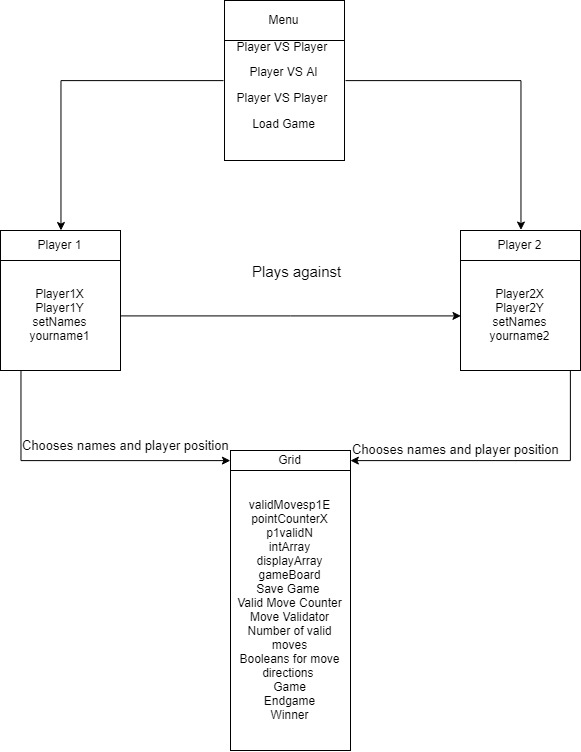
##### 

##### 

##### 

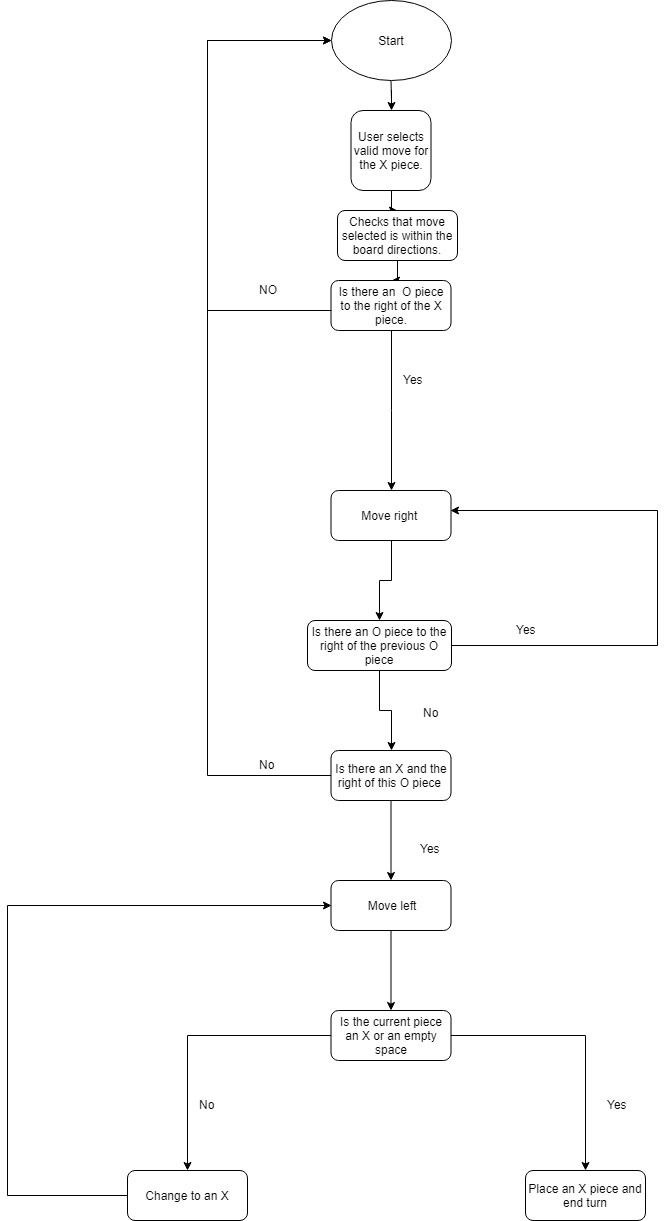
##### 

##### **Class Diagram**

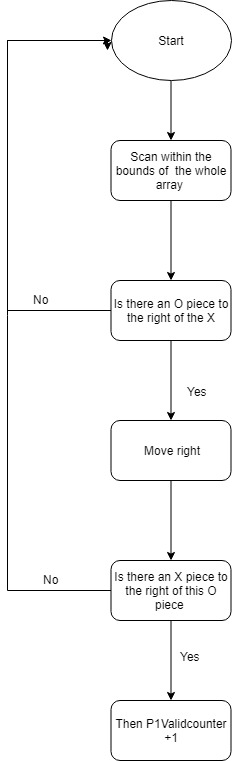


**Flow charts**

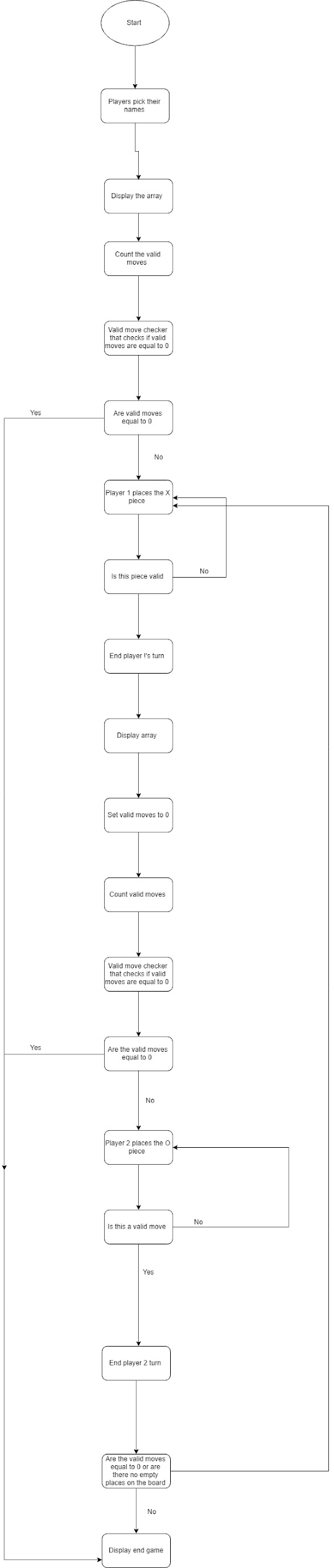
**Method for validating the user input of the X piece on the left.**



**Method for counting the valid moves**

****

**Method for playing the game:**

****

**User Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Playing a game of reverse** |  |  |  |
| **1** | User: | Starts program | Alternatives |
| **2** | System: | Shows player the start menu | A |
| **3** | User: | Chooses Player vs player |  |
| **4** | System: | Asks the player to enter their name and tells them the piece they are gonna be |  |
| **5** | System: | Verifies the users request and starts a new game with an 8x8 board and 4 pieces in the center. . |  |
| **6** | User: | User selects the position to place a piece down. | B |
| **7** | System: | The system verifies users move and updates board then switches to other users turn. |  |
| **8** | User: | Win condition is eventually met |  |
| **9** | System | Count all present pieces between the users. |  |
| **10** | System | Displays a win screen for the user that has the most pieces. |  |
| **11** | System | Player has option to start a new game or exit game | D |
| **12** | System | User selects start new game | Return to step 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **A- User selects load game** |  |  | **Alternatives** |
| **1** | System: | Shows player the start menu |  |
| **2** | User: | Chooses the option load game | c |
| **3** | System: | System loads game |  |
| **4** | User: | User continues of where they were previously playing |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **B- System failing to validate the user’s move.** |  |  |  |
| **1** | System | Displays the game board |  |
| **2** | User: | User places a piece down that doesn’t meet the requirements. |  |
| **3** | System: | System fails verifies users move and reprints the board |  |
| **4** | System | Repeat to step one. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **C- System fails to load game** |  |  |  |
| **1** | User: | Chooses load game |  |
| **2** | System: | Displays the error message no valid save file has been found |  |
| **6** | System: | Repeat to step 1 in the main flow. |  |

**Self-evaluation**

This program was quite the challenge for me, it was the biggest programming project that I had ever done in my life. When starting this program, it was a bit overwhelming since I didn’t know where to start, for a few days I procrastinated and did small parts and when I hit a road block I stopped. After some time, I decided to pick the project back up again. In order to handle the challenges of the program and the overall size I decided to split the program into smaller parts. I split it into Table display, Player input, Menu, save game and load game method, and Move validation.

**Strengths:**

When working on my program I was met with several strengths and weaknesses, the strengths of my program were met when I was working on the overall game method, I decided to make my game method a nested while loop. My main reason was that each player turn will be a while loop, and when ever a move is verified it will make the Boolean for it false ending the loop and avoiding any errors. The large while loop will run continuously until the end of the game is reached.

**Weakness:**

A weakness in my program has to do with the methods that validate the user inputs. These methods originally went through 3 different stages, at first all the directions were in the same method, but this caused an issue with the program since when the player move was placed in a spot that flipped pieces in two different direction then it would cause errors in the program. After I had tried to make these methods worked, I then decided to split each direction into its own individual method. I thought that this would not only reduce the errors that I was experiencing, but also make it easier for me to find what errors was happening in the program. Although this was not the case, even after I changed this aspect of the program the problems persisted. My last resort was to scrap the method completely and start over again, this led to my realization of instead of having the validation of the move and flipping in different methods, what I could do was contain the flipping direction and validation in the same method. This would not only make my program shorter, but also more efficient. Even though I finally got the method to work it did give me quite a bit of trouble since I had to deal with the out of boundary errors that the methods often caused because they would constantly go off the board.

**Parts of my program that made me proud:**

The part of my program that makes me proud is the method that handles the validation of the user inputs and flips them. Even though this method gave me a lot of stress due to the extensive coding that I had to put in for it to finally work, I am happy with the product because when I was done, I managed to create a method that handles the placing validation and flipping of user moves all in one. I believe that this method made coding more efficient since the amalgamation of all these three important functions in one method for each direction made it easy to find the errors in it that eventually were presented in my code. Another aspect of my program that I am happy with is my aspect of the save game method. In my program, rather than the user choosing when to save I implemented an auto save for the user, this means that when playing the game vs another user the user won’t have to pause the flow of the game to save. The program will automatically do it showing the user the message when it does it by displaying a message. This saved file is always saved under the same name, so when loading the game, the user won’t have to worry about forgetting the name of their saved game as the program will always know the right file to pick.

**Overall Evaluation:**

In my opinion, I really struggled with this project when starting I hit a lot of road blocks that really discouraged me from trying harder at first, not to mention the abundance of other projects we had during the time did not help things. The main issue that I had was with my validation methods, flipping and going out of bounds on the array. Handling these issues was where the bulk of my programming time went, I constantly had to ask my friends and Ian on the constant problems I had. At first, I found these issues annoying to deal with and I had to constantly ask for help, but as time went on, I managed to figure out and solve the issues on my own. When compared to the work that we had to do in the labs like the assignment’s and practical’s this project was on another level of difficulty. I had to implement what I had learned throughout the whole semester and even further knowledge. Unlike the other assignment’s I could manage to do one or more optional choices, but I could not manage to do that on these ones in this project.