# Name of Project: *“Reversi”*

**Date: 13th September 2018 (Updated 30/11/2018)**

**Student Names:** *Mohammed Ubaid Tokaria, Pui-Hin Vincent Lee, Dominic McGraw & Mark Capewell.* ***(Updated by Pui-Hin Vincent Lee for final project)***

## Statement of requirements

***Assumptions:***

1. The user might not be an expert with this game “REVERSI”.

***Inputs:***

1. Player moves
2. Start the game
3. Select load game
4. End game

***Outputs:***

1. Main menu screen
2. Load game screen
3. New Discs put down
4. Discs flipping
5. Score
6. End game screen including winner

### Requirements

***Functional Requirements***

*[These should deal with the function of the system, i.e. this will be the main body of requirements that clearly specify what requirements the system shall, should or may meet]*

**R. 1.** The game shall have start menu.

**R. 2.** Start menu shall have start option.

**R. 2.1** Start option shall initiates game.

**R. 3.** Shall be player vs player

**R. 4.** Shall allow user to place disc.

**R. 4.1** Shall block all illegal moves

**R 4.2** Shall flip counters according to the move made

**R. 5.** Shall allow users to save current game.

**R. 6.** Shall allow user load previously saved game.

**R. 7.** Shall declare winner when there are no more legal moves available.

**R. 8.** Shall display discs on 8x8 grid

**R. 9.** Shall display 8x8 grid

**R. 10.** Should automatically pass turn to opponent after move

**R. 11.** Should show both players current scores

**R. 12.** Computer opponent should make intelligent & legal moves

**R. 13.** May have links to rules, tutorials

***Non-functional Requirements***

[These relate to non-functional aspects of the system such as u*sability, performance or system hardware constraints (e.g. minimum hardware specification), required software etc.]*

**R. 1.** Shall run on Java Runtime Environment (JRE)

**R. 2.** Should comply with Copyright, Design & Patent Act 1988.

**Rational** Does not use copyrighted material.

**R. 3.** Should be in English.

**R. 4.** Should run on up to date operating system.

**R. 5.** Computer should make move within 3 seconds.

### User Interface

*[Notes or sketches describing the user interface]*

Exit

Reversi

Start New Game

Load Game

Help

***Main Menu***

*Start New Game:* This option starts a brand-new game taking the used to main board screen and starts a new game of Reversi.

*Load Game:* This brings up the file explorer allowing the user to choose the file of the game they wish to continue.

Help: Set of instructions on How to play Reversi.

*Exit:* This closes the programme.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

***Main Game Menu***

Turn: Yours

**Score**

You: 2

PC: 2

Pause Menu

Turn: This displays weather it is your turn of it’s the computer’s turn.

Score: This display the score of both the player and the computer

Pause Menu: This is a button that allows the user to access the pause menu of the game.

**Notes:** When the board is filled up with all 64 counters, then the game will transition to the finish screen.

Help

Pause

Save Game

Return Home

Exit

***Pause Menu***

Save Game: This brings up the file explorer and allows the used to save the game to a location they desire.

Return Home: This allows the user to return to the main menu of the game.

Help: Set of instructions on How to play Reversi.

Exit: This closes the programme.

Reversi

**Score**

You: 48

PC: 16

**You Win!**

Return Home

***Finish Screen***

This menu is displayed when all 64 positions on the board are filled and indicated whether the user has won or lost the game

At the bottom of this screen there is a “Return Home” button allowing the user to return the main menu.

### Use Cases

*[Consider who or what will use the system and how they will interact with it. Give a descriptive sentence for each use case then the USER: SYSTEM dialogue]*

The use of this cases of this piece of software is to allow the user to play a game against the computer.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Reversi Game*** | | | ***Alternatives*** |
| 1 | User | Opens games |  |
| 2 | System | Opens start menu |  |
| 3 | User | Selects ‘Start Game’ | A |
| 4 | System | Opens new game |  |
| 5 | System | Asks user to place disc |  |
| 6 | User1 | Places disc | B |
| 7 | System | Turn over appropriate disc/s |  |
| 8 | System | Update score |  |
| 9 | User2 | Place disc |  |
| 10 | System | Turn over appropriate disc/s |  |
| 11 | System | Update score |  |
| 12 | System | Repeat steps 7-12 until no more legal moves | C |
| 13 | System | Show scores and winner |  |
| 14 | System | Show end game menu |  |
| 15 | User | Selects ‘Ends Game’ |  |
| 16 | System | Shows main menu |  |

|  |  |  |
| --- | --- | --- |
| **A** | | |
| A1 | User | Selects ‘Load Game’ |
| A2 | System | Opens previously saved game |
| A3 | System | Continues from step 5 |
|  |  |  |
| **B** | | |
| B1 | User | Attempts to place disc in illegal position |
| B2 | System | Show message ‘illegal move – try again’ |
| B3 | System | Continue from step 5 |
|  |  |  |
| **C** | | |
| C1 | User | Ends game before the game is finished |
| C2 | System | Saves game |

### Classes

##### Candidate Classes

*[Fill in the table below with ALL the nouns form the statement of requirements. Note that you do not have to limit yourselves to just these nouns. If a more appropriate noun is sensible then use that]*

|  |  |  |
| --- | --- | --- |
| **Candidate Classes** | **Accept / Reject** | **Reason for rejection** |
| Board | Yes |  |
| Grid | No | Same as board |
| Pieces | No | Too vague |
| Game | Yes |  |
| Opponent | No | Too vague |
| computerOpponent | no | Same methods as a normal user |
| Computer | No | Too vague better described as ‘Opponent’ |
| Light | No | Field of discs |
| Dark | No | Field of discs |
| player | Yes |  |
| Menu | yes |  |

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

*[For each class state what it is responsible for and list the fields (with type) and methods (type and parameters)].*

***Class:*** *Player*

This class is responsible for storing the score of the players and the functions they can execute.

|  |  |  |
| --- | --- | --- |
| ***Fields*** | ***Date Type*** | ***Description*** |
| score | Integer | Stores the score of the player. |
| numOfMoves | Integer | Stores number of moves made by the player. |

|  |  |
| --- | --- |
| ***Methods*** | ***Description*** |
| getScore() | Returns score of the player |
| setScore() | Set score of the player |
| incrementMoves() | Used to increment the moves by 1 when ran. |

***Class:*** *Board*

Used to store the disc positions within the board grid.

|  |  |  |
| --- | --- | --- |
| ***Fields*** | ***Date Type*** | ***Description*** |
| grid | 2D Array (int) | Stores the location of the player counters on the board |
| player1 | Player | An object of made from the player class |
| player2 | Player | An object of made from the player class |
| turn | Integer | Store which player turn it is |
| gameActive | Boolean | Store if the game is active |
| AI | Boolean | Stores weather the game is against the PC or not |

|  |  |
| --- | --- |
| ***Methods*** | ***Description*** |
| Board() | Sets default values for board objects (constructor) |
| isGameActive() | Returns weather the game is active or not |
| score() | Returns the score of the player requested |
| isAI() | Returns weather the game is against the pc or not |
| getTurn() | Returns who turn it is. |
| getGrid() | Returns 2d array of the grid |
| setGrid() | Sets the grid |
| setTurn() | Sets the turn |
| validMoves() | Returns an array of the valid moves for a player |
| countersToFlip() | Checks if locations specified has valid counters to flip |
| calculateScore() | Calculates and set score for each player |
| endTurn() | Runs calculateScore() and switches the turn to the next player if they have valid moves. |
| runTurn() | Takes in the desired placement of the counter and checks if it’s valid then places it and flips counter turn if it is valid. |
| flipCounter() | You define the player and counter locations and flips counters for that user |
|  |  |

**Class:** *Menu*

Holds methods for main menu.

|  |  |
| --- | --- |
| ***Methods*** | ***Description*** |
| options() | Displays available options |
| instructions() | Displays instructions |
| menu() | Loads the menu |

***Class:*** *Game*

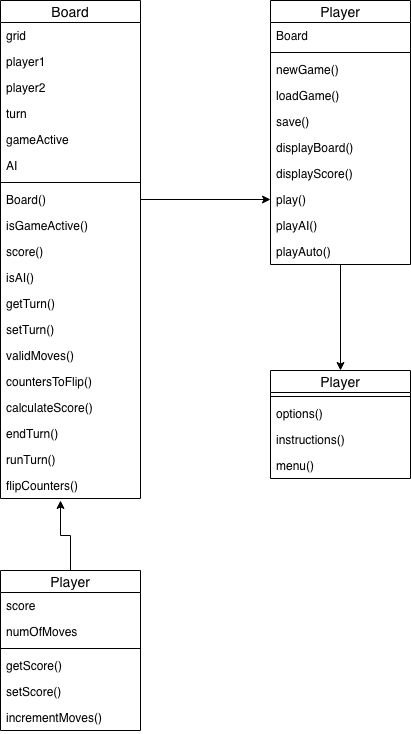
Used to store the board and to interface with it.

|  |  |  |
| --- | --- | --- |
| ***Fields*** | ***Date Type*** | ***Description*** |
| board | Board | Object of Board Class |

Holds methods for main menu & pause menu.

|  |  |
| --- | --- |
| ***Methods*** | ***Description*** |
| newGame() | Displays available options |
| loadGame() | Displays instructions |
| save() | Loads the menu |
| displayBoard() | Displays the board for the CMD view |
| displayScore() | Displays score of each player |
| play() | Takes in user inputs and plays the game by interfacing with the board. |
| playAI() | Takes in user inputs and plays the game by interfacing with the board but also enters the turn with it’s the AI’s go. |
| playAuto() | Used for when a game is loaded in the chose the correct play option. |

**Class Diagram**

****

**Class Descriptions.**

**Menu**

The Menu class is used for displaying the menu, choosing games and displaying the options and instructions.

Methods

* Menu() : *takes in user input and runs appropriate options*
* Options(): *Displays a list of available options*
* Instructions(): *Displays the instructions for the game*

**Player**

This class is used to store parameters that each player will need that track their scores.

Fields

* Score : *score of the player*
* numOfMoves : Store the numbers of moves the player has made.

Methods

* getScore() : *retrieves the score of the player object*
* setScore() : *Sets the score of the player object*
* incrementMoves() : Used to increment the numOfMoves by one when ran()

**Game**

This class is used to take in user input and displaying the game board visually to the user as well as the save and load games by creating board object.

Fields

* board : *An instance of the board class*

Methods

* newGame() : *creates in a brand new game.*
* loadGame() : loads in game from a JSON file and then uses it to assign a board object with the save data.
* save() : *saves the game by saving the board object to a json file.*
* displayBoard() : *displays the board from the board object*
* displayScore() : *displays the score of the players.*
* play() : *Takes in the user inputs when the game is active and end the game with the game becomes inactive.*
* playAI() : *takes in user input and plays the game like play() however when it’s players 2 turn is gets the list of valid moves and chooses a random valid move to input*
* playAuto() : *It chooses whether to run play() or playAI() depending on what the board’s AI field is. This is used for when a game is loaded in.*

**Board**

The board class is used for creating the game itself in terms of the board, where the pieces are, who’s turn it is and the validation and flipping of pieces.

Fields

* grid : *A 2d array that score where the players counters are*
* player1 : *A instance of the player class for player1*
* player 2 : *A instance of the player class for player2*
* turn : Stores whom term it is.
* gameActive : *Used to indicate if the current game is active / Has ended*
* AI : *Used to store whether the game is against an AI for saving the game.*

Methods

* board() : *The constructor method that sets default values*
* isGameActive() : *accessor method of gameActive field*
* score() : *Used for getting the score of the player that is passed through.*
* isAI() : *accessor method of AI field.*
* setAI() : *sets method the AI field.*
* getGrid() : *Returns the 2d array in grid*
* validMoves() : Used for getting a list of valid moves for the player passed to it.
* countersToFlip() : *Used to works out if the players has counters that can be flipped at a defined location on the board.*
* calculateScore() : *Used to calculate the score of each player at the end of turn*
* endTurn() : *This function is ran at the end of a turn and checks if both players have valid moves to determine if the game should end, check whether the next players has any valid moves to make and switches to them if they do.*
* runTurn() : *Takes in the players input and checks if it is a valid input and if it is, checks if it’s valid, if so then places the piece and calls the flip counter method then ends by calling endTurn(). If input isn’t valid it then returns false to the class calling it.*
* flipCounters() : *by passing it a player and cord and it flips the counters that are valid to be flipped.*

***Activity Diagrams / Pseudocode***

*[For any complex or critical methods]*

**Change score**

(Assume human player is playing white)

For all positions on the board

IF position is white

+1 to score

ELSE IF

Don’t increase score

END

**Calculate legal moves**

For all positions on the board

If position is empty

AND position is in line with opponent’s disc/s

AND position is beside opponent’s disc

THEN

Position is valid

Else if Position is invalid

END

**Place disc**

GET position form player

IF position is valid THEN

CREATE new instance of disc

WHERE colour = player colour

AND

ADD disc to grid ARRAY

ELSE IF

SEND error message

END IF