# **Reversi Group Design**

**Date: 14/09/18**

**Student Names:** *Andrew Studders, Calum Logan, Christy Mccarron, Duncan Crampton, Sergej Liubarskij*

## ***Statement of requirements***

**A client requires a functional, interactive computer game version of reversi to be made using the java programming language**

***Assumptions:***

*The player is able to visually interact with their system.*

*The player is over 3 years of age.*

*The player has experience with computers enough to interact with the program.*

*Most players will understand how to play the game after playing an easy mode game.*

***Inputs/Outputs:***

*The player is able to interface with the computer with an input device.*

***Standards:***

*The player’s device will have a Java Runtime Environment installed.*

*The player’s device contains at least 128MB of storage and memory.*

## **Requirements**

**Functional Requirements**

R1. The program shall include a main menu screen. Yes

R2. The program shall include a game screen. Yes

R3. The game screen shall include a grid in an 8x8 format. Yes

R4. The game shall include two colours of pieces; black and white.

Colours represented by B and W

R5. The game should allow the player to choose their colour. Pending

R6. The game shall only permit the player to place a counter in a valid square. In Progress

R7. The game shall designate valid squares as squares next to currently placed pieces. Pending

R8. The game shall designate valid squares as squares which result in the player obtaining at least one enemy piece. Pending

R9. The game shall designate valid squares as squares which connect to at least one other piece of the same colour as the player in a straight line, including diagonally. Pending

R10. The game shall consider a piece captured if it is within two enemy pieces in a straight line. Pending

R11. The game shall change the colour of captured pieces to the colour of the player who made the last move. Pending

R12. The game shall refer to each square in the grid by a number/letter identification e.g. d3. Yes

R13. The game shall begin with 4 pieces already placed on the board: white pieces at d4/e5, black pieces at d5/e4. Yes

R14. The game shall only allow the player with black counters to make the first move. Yes

R15. The game shall end when there are no legal moves left on the board (the board is full). Yes

R16. The game shall announce the winner at the end of the game as the player who possesses the most pieces on an end game screen. Pending

R17. The game should display the score of the winner at the end of the game. Pending

R18. The game should dynamically count the pieces possessed by each player at the end of any move. Yes

R19. The game should display the current score at the end of each move. Yes

R20. The program shall allow two players to play against each other . Yes

R21. The program should allow the player the play against a computer opponent. No

R22. The program may allow the player to select a difficulty to play on against a computer opponent; easy, medium or hard. No

R23. The game should contain the option to save the current game locally. In Progress

R24. The game should contain the option to exit to the menu within a current game. Yes

R25.The program menu shall include an option to start a new game. Yes

R26. The program menu should include an option to load a previously locally saved game. In Progress

R27. The program should save scores from games played to a high-score screen.

In Progress

R28. The program should include an option to see the high-score screen from the main menu. Yes

R29. The program should include an option to view the rules of reverse. In Progress

R30. The game should only allow a player more than one consecutive turn in the case of the enemy player having no legal moves available. Pending

R31. The game should include the option for the player to return to the main menu from the end game screen. Yes

R32. The game may include an ‘undo’ option in easy mode Pending

**Non-functional Requirements**

NFR 1. The game may display squares’ identifications to the player

NFR 2. The game should have a graphical user interface

NFR 3. The game shall not contain flashing graphics

NFR 4. The game should highlight squares where a valid move can be made

NFR 5. The game may include a current tally of wins and losses for the current player

NFR 6. The computer should make its move within 3 seconds

NFR 7. On hard mode, the player should be given a 20 second limit to make their move

NFR 8. The game should be in English

NFR 9. The game’s text should be legible. E.g. fonts that are easy to read

NFR 10. The game shall use no officially licensed logos/artwork

NFR 11. The game should highlight the current player in the GUI

NFR 12. The game may include a log of moves made to display to the player

## **User Interface**

User interface designs made with Photoshop

High score icon obtained from: https://upload.wikimedia.org/wikipedia/commons/thumb/3/3d/Breezeicons-actions-22-games-highscores.svg/500px-Breezeicons-actions-22-games-highscores.svg.png

High score logo obtained from:

<https://www.iconbros.com/icons/ib-se-settings>

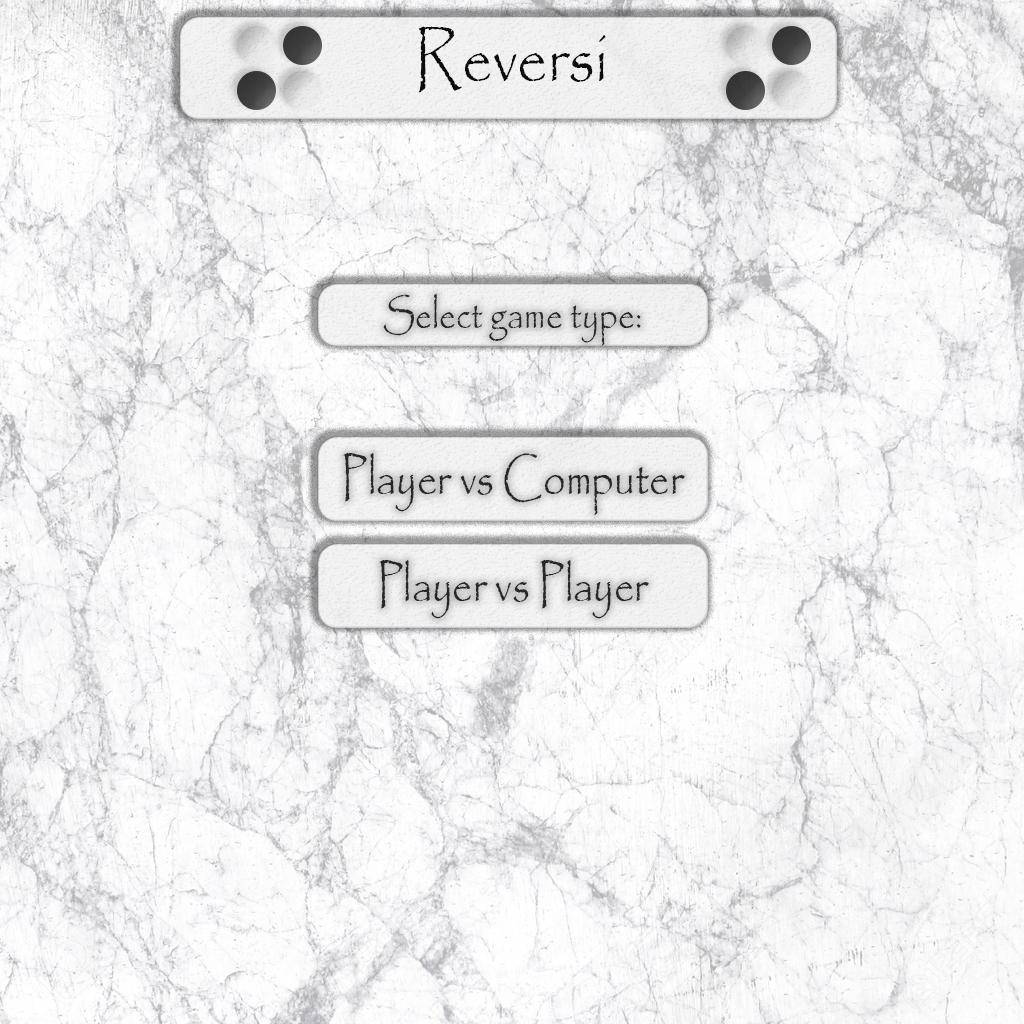
Under open source license. Use is permitted.



*Initial sketch of User Interface*

**

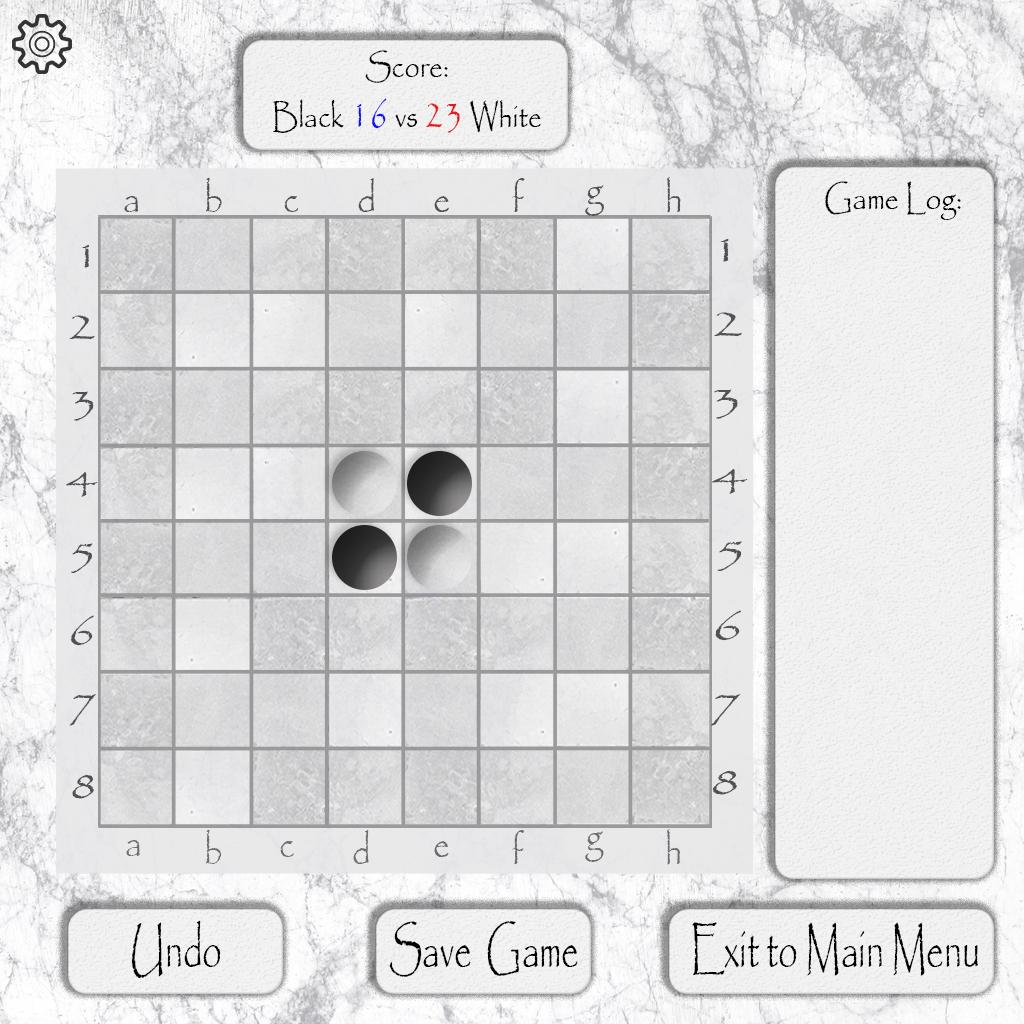
*Fig.1: Main menu interface*

**

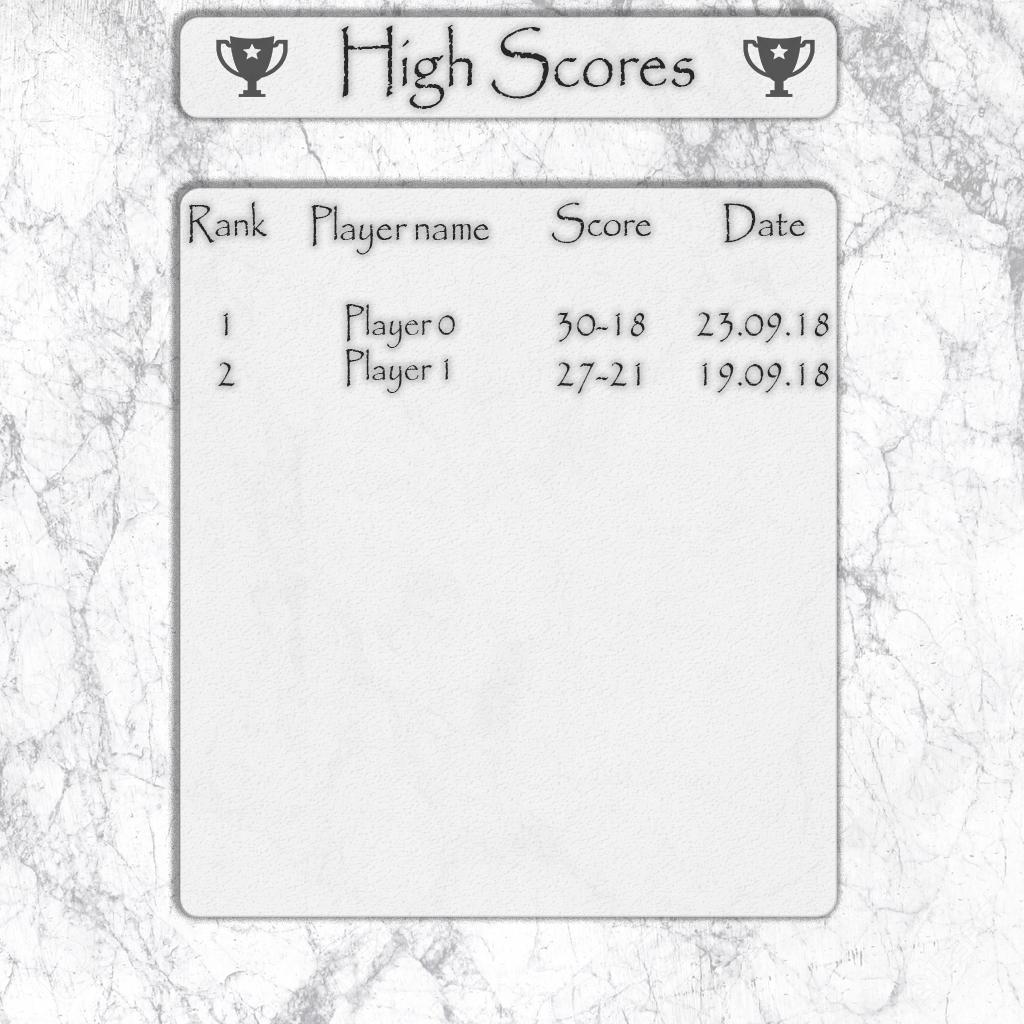
*Fig.2: New game menu*

**

*Fig.3: Player vs. computer difficulty menu*

**

*Fig.4: Game screen*

**

*Fig.5: High Score table*

## **Use Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Step*** | ***UC1*** | ***New Game (main flow)*** | ***Alts*** |
| *1* | *User:* | *Starts program.* |  |
| *2* | *System:* | *Boots up to main menu screen.* |  |
| *3* | *User:* | *Selects ‘new game’ from main menu.* | *AC1*  *AC2*  *AC3* |
| *4* | *System:* | *Prompts User to choose whether to play a player vs. player or player vs. computer game.* |  |
| *5* | *User:* | *Selects ‘Player vs. Computer’.* | *AC4* |
| *6* | *System:* | *Asks user which difficulty of opponent to play against (‘Easy’, ‘Medium’, ‘Hard’).* |  |
| *7* | *User:* | *Selects ‘Easy’.* | *AC5*  *AC6* |
| *8* | *System:* | *Starts a new game against an easy computer opponent with ‘undo button’ functionality (saves the player’s previous move and allows them to undo it when used).* |  |
| *9* | *System:* | *Initializes the playing board with vertical axis labelled A-H and horizontal axis labelled 1-8* |  |
| *10* | *System:* | *Refers to each square by the appropriate letter and number combination* |  |
| *11* | *System:* | *Places black pieces on D5 and E4 and white pieces on D4 and E5* |  |
| *12* | *User:* | *Chooses which colour they wish to plag as* |  |
| *13* | *System* | *Sets the black player to have the first move* |  |
| *14* | *User:* | *Attempts to make move.* | *AC7*  *AC8*  *AC9*  *AC10* |
| *15* | *System:* | *Verifies player has not already made a move this turn.* | *AC11* |
| *16* | *System:* | *Verifies player is able to make a valid move.* | *AC12* |
| *17* | *User:* | *Makes valid move* |  |
| *18* | *System:* | *Identifies the playing pieces that have been captured* |  |
| *19* | *System:* | *Changes the colour of the appropriate playing pieces* |  |
| *20* | *System:* | *Adds the number of captured pieces plus one to the current players score* |  |
| *21* | *System:* | *Subtracts the number of captured pieces from the other players score* |  |
| *22* | *System:* | *Updates the score values on the interface* |  |
| *23* | *System:* | *Changes the game to disallow the user from making another move* |  |
| *24* | *System* | *Saves a copy of the current board layout.* |  |
| *25* | *System:* | *The game meets the end game criteria (no possible move by either party)* |  |
| *26* | *System:* | *Displays message “[winning player] has won with a score of [score]”* | *AC13* |
| *27* | *System:* | *Adds one to the winner’s tally* |  |
| *28* | *System:* | *Updates the values on the interface* |  |
| *29* | *System:* | *Displays menu options* |  |
| *30* | *User:* | *Selects main menu* |  |
| *31* | *System:* | *Closes the game display* |  |
| *32* | *System:* | *Opens the main menu* |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Step*** | ***AC No.*** | ***Alternative Use Cases*** | ***Alt.2*** |
|  | ***AC1*** | ***User loads save game*** |  |
| *1* | *User:* | *Selects load game* |  |
| *2* | *System:* | *Presents user with a list of all saved game in location selected* |  |
| *3* | *User:* | *Selects a saved game from the list* |  |
| *4* | *System:* | *Loads selected game* |  |
| *5* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC2*** | ***User selects ‘View High Score’*** |  |
| *1* | *User:* | *Selects ‘View High Scores’ from main menu* |  |
| *2* | *System:* | *Loads high score screen* |  |
| *3* | *User:* | *Exits high scores screen* |  |
| *4* | *System:* | *Returns to step 2 of main flow* |  |
|  | ***AC3*** | ***User selects ‘View Rules’*** |  |
| *1* | *User:* | *Selects ‘View Rules’ from main menu* |  |
| *2* | *System:* | *Loads rules screen* |  |
| *3* | *User:* | *Exits rules screen* |  |
| *4* | *System:* | *Returns to step 2 of main flow* |  |
|  | ***AC4*** | ***User selects ‘Player vs Player’ instead of ‘Player vs Computer’*** |  |
| *1* | *System:* | *Loads a new game for 2 players* |  |
| *2* | *System:* | *Returns to step 9 in main flow* |  |
|  | ***AC5*** | ***User chooses medium opponent*** |  |
| *1* | *User:* | *Selects ‘medium’ difficulty of computer to play against* |  |
| *2* | *System:* | *Load new player vs computer game with medium difficulty* |  |
| *3* | *System* | *On medium difficulty the system will select the second best move from the avalibible moves* |  |
| *4* | *System:* | *Returns to step 9 of main menu* |  |
|  | ***AC6*** | ***User chooses hard opponent*** |  |
| *1* | *User:* | *Selects ‘hard difficulty’ to play against* |  |
| *2* | *System:* | *Loads new player vs computer game with hard difficulty* |  |
| *3* | *System:* | *On ‘hard difficulty’ the system will select the best move from the available moves* |  |
| *4* | *System:* | *Returns to step 9 of main flow* |  |
|  | ***AC7*** | ***Player fails to make a move during the time limit*** |  |
| *1* | *System:* | *Verifies player failed to make a move* |  |
| *2* | *System:* | *Displays error message to user* |  |
| *3* | *System:* | *Allows the other player to make a move* |  |
| *4* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC8*** | ***User selects save game option*** |  |
| *1* | *System:* | *Prompts user to select location to save the game locally* |  |
| *2* | *User:* | *Inputs location to save the game to* |  |
| *3* | *System:* | *Verifies location is valid for local systems* | *AC8.1* |
| *4* | *System:* | *Prompts user to input filename of game* |  |
| *5* | *User:* | *Inputs desired filename* |  |
| *6* | *System:* | *Verifies filename contains no forbidden characters e.g. \* . “ / \ [ ] : ;* | *AC8.2* |
| *7* | *System:* | *Writes txt file of current game to location specified as filename* |  |
| *8* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC8.1*** | ***User has entered an invalid location*** |  |
| *1* | *System:* | *Can’t find location entered* |  |
| *2* | *System:* | *Displays error message to user* |  |
| *3* | *System:* | *Returns to Step 1 In AC8* |  |
|  | ***AC8.2*** | ***User has entered a name with illegal characters*** |  |
| *1* | *System:* | *Finds illegal characters in filename given* |  |
| *2* | *System:* | *Displays error message to user* |  |
| *3* | *System:* | *Returns to step 4 in AC8* |  |
|  | ***AC9*** | ***User selects Exit option*** |  |
| *1* | *System:* | *Prompts user to confirm their exit, save before exit, or cancel exit* | *AC9.1*  *AC9.2* |
| *2* | *User:* | *Confirms their exit* |  |
| *3* | *System:* | *Returns program to main menu* |  |
|  | ***AC9.1*** | ***User chooses to cancel their exit*** |  |
| *1* | *User:* | *Selects ‘Cancel Exit’* |  |
| *2* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC9.2*** | ***User saves before exit*** |  |
| *1* | *System:* | *Go to AC8* |  |
| *2* | *User:* | *Confirms their exit* |  |
| *3* | *System:* | *Returns program to main menu* |  |
|  | ***AC10*** | ***The user selects the undo option in ‘easy mode’*** |  |
| *1* | *System:* | *The system moves the saved copy of playing board onto the playing board* |  |
| *2* | *System:* | *The undo button is disabled until the user makes a move* | *AC10.1* |
| *3* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC10.1*** | ***The user tries to press the undo button multiple times*** |  |
| *1* | *System:* | *Displays error message to user* |  |
| *2* | *System* | *Returns to step 2 in AC10* |  |
|  | ***AC11*** | ***The player has already made a previous move this turn*** |  |
| *1* | *System:* | *Verifies the player has already made a move* |  |
| *2* | *System:* | *Displays error message to user* |  |
| *3* | *System:* | *Allows the other player to make a move* |  |
| *4* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC12*** | ***Player is no longer able to make any valid moves*** |  |
| *1* | *System:* | *Verifies player can’t make any more moves* |  |
| *2* | *System:* | *Displays error message to user* |  |
| *3* | *System:* | *Allows the other player to make a move* |  |
| *4* | *System:* | *Returns to step 13 in main flow* |  |
|  | ***AC13*** | ***Game ends in a draw*** |  |
| *1* | *System:* | *Verifies players have the same score* |  |
| *2* | *System:* | *Displays message “The game has ended in a draw”* |  |
| *3* | *System:* | *Returns to step 28 in main flow* |  |

## **Class Descriptions**

***User :*** *Contains information about Human Players of the game.*

***Fields***

* *Username - string of the current user*
* *SavedGames - list of arrays of the User’s saved games*
* *Tallies - wins/losses vs players and vs computer*

***Methods***

* *New User - adds a new user to the program*
* *Load User Data - loads all previous user data from file into the program*
* *Get Saved Games - returns all the users saved games s*
* *Get Tally - returns the tally of wins or losses*

***Game -*** *Contains all unique information relevant to the current game, e.g. difficulty and type, and sets up a new game.*

***Fields***

* *Score - stores the score from the current game*
* *Current Game Mode - from 1-4 ,1 being player vs player 2-4 being easy to hard*

***Methods***

* *Check Valid Moves - calculate all valid moves that can be made*
* *Check End Game - checks if the game has been completed(no player has a move)*
* *Undo - copies the data from temp board(last move) into the current board*
* *Player Vs Computer - the method that runs the difficulty selection for the ai mode*
* *Easy - the method that runs the game in easy mode(random move, unlimited time, highlighted available moves)*
* *Medium - the method that runs the game in medium mode(2nd best move, unlimited time, no highlighted moves)*
* *Hard - the method that runs the game in hard mode (best move, 20 second time limit, no highlighted moves)*
* *Player vs Player - the method that runs the game in 2 player mode*
* *Save Game - Saves the current game to the User Object and saves the User Data Locally*

***Board*** *- The class used for board setup and recording the current state of board after each move*

***Fields***

* *Grid - 2d array*
* *Log - list of array - holds the move data*

***Methods***

* *StartNewBoard - sets up the board for a new game ( counters in middle)*
* *Move - after the system or player makes a legal move has made a move this method calculates what pieces have been captured, also saves this data to the log*
* *Flipper - after move has been called this method will flip the captured pieces*

***Objects***

* *CurrentBoard - the object responsible for holding the data on the board for the current game*
* *TempBoard - the object responsible for holding the data on the previous move, used in the undo method of the game class*

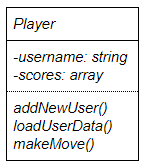
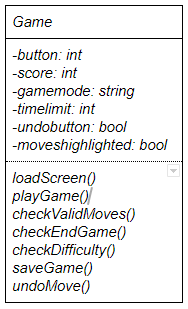
***Counter-*** *The class for holding the counter data*

***Fields***

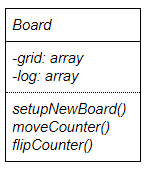
* *Colour - Stores the colour data of the counter*

***Methods***

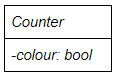
## **Class Diagram**



Plays



Contains



Contains

**Activity Diagrams / Pseudocode**

Diagrams and Pseudocode made with Code Rocket Designer

[*https://www.rapidqualitysystems.com/Products/CodeRocketDesigner*](https://www.rapidqualitysystems.com/Products/CodeRocketDesigner)

|  |  |
| --- | --- |
|  | **Setting up a new board (using methods 2 and 3 below)**  **private** void setupNewBoard()  {  //setup 8x8 grid  // initialize grid  // set d4 and e5 to white counters  // set e4 and d5 to black counters  } |

|  |  |
| --- | --- |
|  | **Initializes the board to be accessed by other methods**  **private** void setup\_and\_initialize()  {  // Create 8x8 2d array  // For x = 0 to length of board -1 ,x iterate by 1  **for**()  {  // For y = 0 to width of board -1 ,y iterate by 1  **for**()  {  // Board x,y position is assigned to \*  }  }  } |

|  |  |
| --- | --- |
|  | **Adding initial counters to board**  **private** void add\_Counters()  {  // board 4,4 and 5,5 set to white  // board 4,5 and 5,4 set to black  } |

|  |
| --- |
|  |
| **Changing to a different screen (e.g. game to menu, menu to high scores)**  **private** void loadScreen()  {  // Switch Screen  **switch**()  {  // 1: Game Screen  **case**:  // close MainMenu  // close HighScores  // open ChooseModeScreen  // Which Mode?  **switch**()  {  // Player vs Computer  **case**:  // Choose Difficulty  **switch**()  {  // Easy  **case**:  // open GameScreen  // Medium  **case**:  // open GameScreen  //Hard  **case**:  // open GameScreen  }  // Player vs Player  **case**:  // open GameScreen  }  // 2: HighScores  **case**:  // close MainMenu  // close GameScreen  // Open HighScores screen  // 3: Main Menu  **case**:  // close GameScreen  // close HighScores  // Open MainMenu  }  } |

|  |  |
| --- | --- |
|  | **Setting up a new game from menu**  **private** void setupGame()  {  //Call setupNewBoard  // If Player vs Computer  **if**()  {  // Assign colours to players  //Call checkDifficulty  // Return 2-4  }  //Else  **else**  {  //Assign colours to players  //Call checkValidMoves  // Return 1  }  } |

|  |
| --- |
|  |
| **Main method for logic of playing a game**  **private** void PlayGame()  {  // If Call setupGame() =1  **if**()  {  // Repeat until game is over  **do**  {  //Call checkValidMoves  // player makes move  //Call moveCounter  //Call flipCounter  // change player  //Call checkEndGame  }  **while**();  }  //Else  **else**  {  //Call checkDifficulty  // set computer difficulty  // Repeat until game is over  **do**  {  //Call checkValidMoves  // player makes move  //Call moveCounter  //Call flipCounter  // change player  //Call checkEndGame  }  **while**();  }  } |

|  |
| --- |
| **Checking if a player’s move is valid**    **private** void checkValidMoves()  {  // set boolean allFound = false  // 'find all edge pieces'  // create ValidMove[][] 2d array  // While ( allFound is false) (!allfound)  **while**()  {  // for int x = 0, until length of board -1  **for**()  {  // for int y = 0 , until length of board -1  **for**()  {  //if board[x][y] = otherPlayersPiece  **if**()  {  // int candY = y  // int candX = x  // Repeat until foundLeft = true or hit edge  **do**  {  // check to left  // If find players piece  **if**()  {  // bool foundLeft = true  }  //Else  **else**  {  // check if hit edge, then break  }  }  **while**();  // If foundLeft is true  **if**()  {  // validMove[x] = x  // validMove[y] = y  // validMoveFound = true  }  //Else  **else**  {  }  // check right(same as left)  // check above(same as left)  // check below(same as left)  }  // if validMoveFound is true  **if**()  {  // allFound = true  }  //Else  **else**  {  // return no valid move  // break  }  }  }  }  } |