6 Men Morris

Design Document

COMPSCI 2ME3/ SFWRENG 2AA4

Group 19

Nathan Mangaoil

Yikern Looi

Arian Sohrabi

**Description of classes and modules**

In this project consist of 4 java files, “GameBoard.java”, “GameLogic.java”, “UserInterface.java” and “AIControl.java”. These files are the decomposition modules of the game “6 men Morris”. Decomposing the game into this 4 Java files provide a spread in focus, where we have GameBoard and partially UserInterface to display front end of the game; whereas GameLogic focuses on game mechanics and rules; AIControl gives the ability to play against a bot. It make sense to split the files that focus primarily the front end and back end of the game.

**GameBoard.Java**

This Java class extends JPanel to compensate for graphic methods.

* *paintComponent (Graphics g)*
  + Creates a display of the game board on screen.
* *drawLine( int(arg 0), int (arg 1), int(arg 2), int (arg 3))*
  + This creates a line on board using 2 points.
* *SetColor (Color.arg 0)*
  + This paints the respective color requested by the user.
* *fillOval(x, y, int(height), int(length))*
  + This is used to create templates for the chips to be placed on.

**GameLogic.Java**

This module that implements the validation of the board, rules and chop placed in a mill position.

* *countPoints()*
  + Checks the board for how many points for each player
  + Uses
    - *Private static int bluePoints*
    - *Private static int redPoints*
* *checkRow()*
  + Checks the rows on the board
  + Uses
    - *Private static Color[]*
* *checkCol()*
  + Checks the columns on the board
  + Uses
    - *Private static Color[]*
* *updatePoints(Color [] newpoints)*
  + Updates points on the board
  + Uses
    - *Private static Color[]*
* *addPiece(Color player)*
  + This method separates the playter with Blue and Red chips to be placed
  + Uses
    - *Private static int bluePlaced*
    - *Private static int redPlaced*
* *checkPhase()*
  + Method that checks the status of the game and decides when to switch game phase
  + Uses
    - *Private static int bluePlaced*
    - *Private static int redPlaced*
* *isValid()*
  + Boolean method that sees if the current board state is valued
  + Uses
    - *Private static int bluePoints*
    - *Private static int redPoints*
* *getBlue()*
  + Method that returns blue points
  + Uses
    - *Private static int bluePoints*
* *getRed()*
  + Method that returns red points
  + Uses
    - *Private static int redPoints*
* *setPlaced (int blue, int red)*
  + Method that allows the user to the set bluePlace and redPlaced values
  + Uses
    - *Private static int bluePoints*
    - *Private static int redPoints*
* *isMoveable(int i)*
  + method that checks if a piece can be moved
  + Uses
    - *Private static Color[]*
* *validMove(int a, int b)*
  + Method that checks if 2 spaces are adjacent and if it’s possible to move set piece
  + Uses
    - *Private static Color[]*

**UserInterface.Java**

Main class module that controls the game. It focuses on the player’s freedom to place the chip anywhere using a mouse.

* *setButtons()*
  + Method that adds buttons to a JPanel which is implemented
* *setScorePane()*
  + Adds score information to a JPanel
* *setScores()*
  + Sets up score labels on to a JPanel
* *Handlerclass*
  + This class handles all the *Abstract Window Toolkit(AWT),* mouse events
  + Implements
    - *mouseDragged(MouseEvent e)*
    - *mouseMoved(MouseEvent e)*
    - *mouseEntered(MouseEvent e)*
    - *mouseExited(MouseEvent e)*
    - *mousePressed(MouseEvent e)*
    - *mouseReleased(MouseEvent e)*
    - *actionPerformed(ActionEvent e)*
* *choosePlayer()*
  + Method that decides which player goes first
* *updateScores()*
  + Method that updates the score labels and checks for wins
* *AIMill(int x)*
  + This method is used to check mills for the bot
* *AITurn()*
  + Method that is used whenever that AI has to make a move

AIControl.Java

Main class module that controls the game, this module focuses on what the bot will do on the next move. Giving the user the ability to play against a bot in the game.

* *placePiece(Color[] board)*
  + The bot pick’s a spot on the board to place its colored chip.
* *removePiece(Color[] board, Color playerPiece)*
  + The bot removes the player’s chip when the bot achieves mill.
* *pickPiece(Color [] board, Color aiPiece, GameLogic logic)*
  + The bot selects its own piece so that it can be moved.
* *movePiece(Color [] board, Color aiPiece, GameLogic logic, int currentSpot)*
  + The bot selects its targeted spot to be moved

Breaking down into these components of “GameBoard.Java”, “Gamelogic.Java”, “UserInterface” and “AIControl.Java”. Furthermore, into a lot of classes and method because by making it easier to read for the programmer to be able to pinpoint errors when they occur.

**Interface documentation**

|  |  |  |
| --- | --- | --- |
| GameBoard | | |
| Classes | | Method Name | | | Method Type | | Passed Arguments | | Return Data | | Comments | |
| GameBoard | | paintComponent | | | Public | | Graphics g | | graphics | | Paints the JPanel with graphics | |
| GameBoard | | | Public | | Color[] colors | | void | | Sorts the colors of the dots | |
| GameLogic | | |
| Classes | Method Name | | | Method Type | | Passed Arguments | | Return Data | | Comments | |
| GameLogic | countPoints | | | Public | | - | | Void | | Counts Points on the board by iterating through the board | |
| checkRow1 | | | Public | | - | | Void | | Checks the first row for mills | |
| checkRow2 | | | Public | | - | | Void | | Checks the second row for mills | |
| checkRow3 | | | Public | | - | | Void | | Checks the third row for mills | |
| checkRow4 | | | Public | | - | | void | | Checks the fourth row for mills | |
| checkCol1 | | | Public | | - | | Void | | Checks the first column for mills | |
| checkCol2 | | | Public | | - | | Void | | Checks the second column for mills | |
| checkCol3 | | | Public | | - | | Void | | Checks the third column for mills | |
| checkCol4 | | | Public | | - | | Void | | Checks the fourth column for mills | |
| isValid | | | Public | | - | | Boolean | | Checks the current state of the board if its valid | |
| getBlue | | | Public | | - | | Int | | Gets points for player blue | |
| getRed | | | Public | | - | | Int | | Get points for player red | |
| updatePoints | | | Public | | Color[] newpoints | | Int | | Updates the color array for the game logic | |
| addPiece | | | Public | | Color player | | Int | | To increment the blue placed and redPlaced counters whenever | |
| checkPhase | | | Public | | - | | Int | | Checks the status of the game and decides when to switch game phases | |
| setPlace | | | Public | | Int blue  Int red | | Int | | to the set bluePlace and redPlaced values | |
| isMoveable | | | Public | | Int i | | Boolean | | checks if a piece can be moved | |
| validMove | | | Public | | Int a  Int b | | Boolean | | Check if 2 spaces are adjacent and if it’s possible for a move | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| UserInterface | | | | | |
| Classes | Method used | Method Type | Passed Arguments | Return Data | Comments |
| UserInterface | setButtons | Private | - | JPanel | Creates a JPanel that holds buttons and labels |
| UserInterface | Private | - | JPanel | Display a small window titled “Six-men Morris” |
| setScorePane | Private | - | JPanel | Sets scores for the players |
| setScores | Private | - | JPanel | Sets up score labels |
| choosePlayer | Private | - | Void | Decides which player goes first |
| Handlerclass | mouseClicked | Public | MouseEvent e | Void | This gives the player the freedom to add the chips |
| mouseDragged | Public | MouseEvent e | Void | Mouse drag action |
| mouseMoved | Public | MouseEvent e | Void | Mouse move action |
| mouseEntered | Public | MouseEvent e | Void | Mouse entered action |
| mouseExited | Public | MouseEvent e | Void | Mouse exited action |
| mousePressed | Public | MouseEvent e | void | Mouse pressed action |
| actionPerformed | Public | ActionEvent e | void | This method listens for mouse events and implements it |
| AITurn | Private | - | JPanel | Whenever that AI needs to make a move |
| AIMill | Private | int | JPanel | This Method checks whether if the AI has a mill and proceed to the method AITurn |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| AIControl | | | | | |
| Classes | Method used | Method Type | Passed Arguments | Return Data | Comments |
| AIControl | placePiece | Public | Color[] board | Int | Pick a spot on the board and return that spot if it's empty |
| removePiece | Public | Color[] board  Color Piece | Int | Pick a spot on the board and return that spot if it has the other player's piece |
| pickPiece | Public | Color[] board  Color Piece  GameLogic logic | Int | Pick a spot on the board and return that spot if it has the other player's piece |
| movePiece | Public | Color[] board  Color Piece  GameLogic logic  Int | Int | Pick a spot on the board and return that spot if it the AI can move it's piece there |

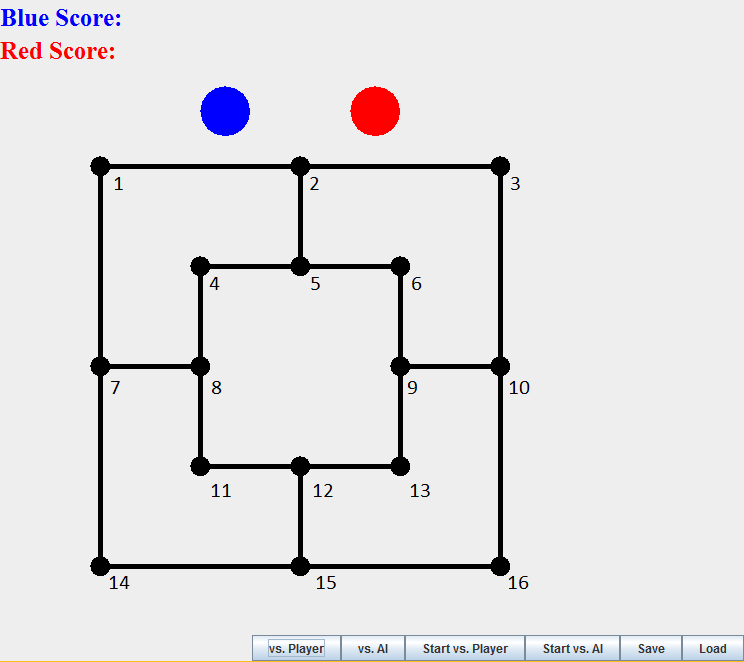
**Uses of Relationship**

Below displays a table that shows the uses of relation amongst the modules. The relation is viewed from left to right that forms a hierarchy.

|  |  |  |  |
| --- | --- | --- | --- |
| Relation | | | |
| JFrame | JPanel | GameBoard | Board |
| GameLogic | Logic |
| countPoints |
| checkPhase |
| validMove |
| setPlace |
| isMoveable |
| JLabel | Status |
| Valid |
| blueScores |
| redScores |
| JButton | Vs Player |
| Save |
| Load |
| Vs AI |
| Start vs Player |
| Start Vs AI |
| Handlerclass | Handler |
| mouseClicked |
| actionPerformed |
| AIMill |
| AITurn |
| setScorePane | setScores |
| AIControl | placePiece |
| removePiece |
| pickPiece |
| movePiece |

We currently use this relations to direct all the methods and transfer the data onto JFrame, hence to be able to play the game.

**Behaviour table**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition | | | | Results |
| Chip Slot | X < 90 | | | Invalid Position |
| 90 <= x <= 110 | | Y < 90 | Invalid Position |
| 90 <= y <= 110 | Paints colour |
| 110 < y | Invalid Position |
| 110 < x | | | Invalid Position |
| X < 290 | | | Invalid Position |
| 290 <= x <= 310 | | Y < 90 | Invalid Position |
| 90 <= y <= 110 | Paints colour |
| 110 < y | Invalid Position |
| 310 < x | | | Invalid Position |
| X < 490 | | | Invalid Position |
| 490 <= x <= 510 | | Y < 90 | Invalid Position |
| 90 <= y <= 110 | Paints colour |
| 110 < y | Invalid Position |
| 510 < x | | | Invalid Position |
| X < 190 | | | Invalid Position |
| 190 <= x <= 210 | | Y < 190 | Invalid Position |
| 190 <= y <= 210 | Paints colour |
| 210 < y | Invalid Position |
| 210 < x | | | Invalid Position |
| X < 290 | | | Invalid Position |
| 290 <= x <= 310 | | Y < 190 | Invalid Position |
| 190 <= y <= 210 | Paints colour |
| 210 < y | Invalid Position |
| 310 < x | | | Invalid Position |
| X < 390 | | | Invalid Position |
| 390 <= x <= 410 | | Y < 190 | Invalid Position |
| 190 <= y <= 210 | Paints colour |
| 210 < y | Invalid Position |
| 410 < x | | | Invalid Position |
| X < 90 | | | Invalid Position |
| 90 <= x <= 110 | | Y < 290 | Invalid Position |
| 290 <= y <= 310 | Paints colour |
| 310 < y | Invalid Position |
| 110 < x | | | Invalid Position |
| X < 190 | | | Invalid Position |
| 190 <= x <= 210 | | Y < 290 | Invalid Position |
| 290 <= y <= 310 | Paints colour |
| 310 < y | Invalid Position |
| 210 < x | | | Invalid Position |
| X < 390 | | | Invalid Position |
| 390 <= x <= 410 | | Y < 290 | Invalid Position |
| 290 <= y <= 310 | Paints colour |
| 310 < y | Invalid Position |
| 410 < x | | | Invalid Position |
| X < 490 | | | Invalid Position |
| 490 <= x <= 510 | | Y < 290 | Invalid Position |
| 290 <= y <= 310 | Paints colour |
| 310 < y | Invalid Position |
| 510 < x | | | Invalid Position |
| X < 190 | | | Invalid Position |
| 190 <= x <= 210 | | Y < 390 | Invalid Position |
| 390 <= y <= 410 | Paints colour |
| 410 < y | Invalid Position |
| 210 < x | | | Invalid Position |
| X < 290 | | | Invalid Position |
| 290 <= x <= 310 | | Y < 390 | Invalid Position |
| 390 <= y <= 410 | Paints colour |
| 410 < y | Invalid Position |
| 310 < x | | | Invalid Position |
| X < 390 | | | Invalid Position |
| 390 <= x <= 410 | | Y < 390 | Invalid Position |
| 390 <= y <= 410 | Paints colour |
| 410 < y | Invalid Position |
| 410 < x | | | Invalid Position |
| X < 90 | | | Invalid Position |
| 90 <= x <= 110 | | Y < 490 | Invalid Position |
| 490 <= y <= 510 | Paints colour |
| 510 < y | Invalid Position |
| 110 < x | | | Invalid Position |
| X < 290 | | | Invalid Position |
| 290 <= x <= 310 | | Y < 490 | Invalid Position |
| 490 <= y <= 510 | Paints colour |
| 510 < y | Invalid Position |
| 310 < x | | | Invalid Position |
| X < 490 | | | Invalid Position |
| 490 <= x <= 510 | | Y < 490 | Invalid Position |
| 490 <= y <= 510 | Paints colour |
| 510 < y | Invalid Position |
| 510 < x | | | Invalid Position |
| Vs Player | Pressed within domain | | | Randomized player to start placing chips |
| Pressed outside domain | | | Invalid Position |
| Vs AI | Pressed within domain | Bot’s turn | | Randomly place chip in any of the spot |
| Player’s turn | | Player to place chip in any spot |
| Pressed outside domain | | | Invalid Position |
| Start Button | Pressed within domain | Less than 6 red and blue chips on board | | Board is invalid |
| 6 red chips and 6 blue chips on board | | Board is valid, initiates’ game.  Game is in progress |
| More than 6 red and blue chips on board | | Board is invalid |
| Either red chips more than blue chips on board | | Board is invalid |
| Pressed outside domain | | | Invalid Position |
| Mills | Board is valid | Chips placed in threes   * 1,2,3 * 1,7,14 * 3,10,16 * 4,5,6 * 4,8,11 * 11,12,13 * 6,9,13 * 14,15,16 | | Board is in a mill position which displays Mill, allowing player to remove opposing player’s chip |
| Chips placed not in a form of mill | | Game is in progress |
| Board is invalid | Chips placed in threes   * 1,2,3 * 1,7,14 * 3,10,16 * 4,5,6 * 4,8,11 * 11,12,13 * 6,9,13 * 14,15,16 | | Board is invalid |
| Chips placed not in a form of mill | | Board is invalid |
| Saving | Pressed within the domain | Game is saved | | |
| Pressed outside the domain | Invalid move | | |
| Loading | Pressed within the domain | Game is loaded from previous save | | |
| Pressed outside the domain | Invalid move | | |

**Implementation Description**

|  |  |
| --- | --- |
| Requirements | Classes   * Methods that uses to solve |
| The board should include two different kinds of discs (blue and red), one for each player, initially placed on either side of an empty board. | **Gameboard**   * paintComponent   **Handlerclass**   * mouseClicked * actionPerformed |
| The order of play (blue first or red first) shall be determined randomly | **UserInterface**   * choosePlayer |
| The user shall be able to choose to start a new game, or enter discs to represent the current state of a game by placing different coloured discs in the frame | **UserInterface**   * setButtons   **Handlerclass**   * mouseClicked * actionPerformed |
| system should analyze whether the current state is possible or not | **GameLogic**   * isValid |
| all the errors shall be highlighted in some way on the screen | **Handlerclass**   * actionPerformed |
| The results of the game must be displayed at all times | **Handlerclass**   * mouseClicked |
| Storing an existing unfinished game | **Handlerclass**   * actionPerformed |
| Restart a stored game | **Handlerclass**   * actionPerformed |
| Being able to choose between two modes of operation: 2 player Six Men’s Morris, in which 2 people can play against each other; or 1 player against the computer | **Handlerclass**   * actionPerformed   **AIControl**   * placePiece * removePiece * pickPiece * movePiece |
| You will need an algorithm to determine the best next move for the computer | **Handlerclass**   * actionPerformed   **AIControl**   * placePiece * removePiece * pickPiece * movePiece |

Variables are maintain in an array of bounded pixels inside JFrame. Each array is a slot on the game that stores a color, each of the colours can be repaint. The method *paintComponent(Graphics g)* draws the line and slots for the boundaries of the game.

In the **Handlerclass,** this class provides full user control through a mouse. Mouse action methods such as *mousePressed, mouseMoved, mouseReleased, mouseEntered* are all taken into account to provide a friendly user interaction with the game.

Furthermore in **Handlerclass**, the class also provides the ability to save the game onto a text file. In that class, also able to read the save files and load the same game. In *actionPerformed* method in the **Handlerclass** has given a simple AI program to play the game, providing it methods and loops to pick their next action performed.

**Internal Review**

We are pleased that our product is completed, fit with the option to play against a bot. In Assignment 3 we do not want to tear apart our code to compensate Ai capabilities. Instead we created a new decomposition module just for AI controls, using this method gives the program to consistently make moves when needed.

In our previous assignment we struggled with Rules, boundaries, save and loading files. As of assignment 1 we have refine our algorithm to fit Ai capabilities into the game. This assignment gave us difficulty alter the code to fit AI algorithm without breaking the code down, to solve this the team decides to try one method at a time instead of putting in the codes without testing and hoping it will run smoothly.

In our second assignment, one of our problems was trying to make the game work within the rules and boundaries. Using the template from the first assignment, we able to discuss a way to solve the problem by solving small task at the time. We started off by solving if the game knows how to recognize mills and able to remove a chip from the other player.

In our first assignment where users have the freedom of selecting the slot and we have achieved it. This time, to develop the necessary game mechanics to satisfy the rules. Our initial thinking of decomposing the game into more modules, but making the game work by recognizing mills and able to reload a save game was our second on the list.

We have thought of just creating the methods within **Handlerclass** to avoid confusion, but we were wrong, by doing so we consistently crash the program and unable to run. By solving the problem we use another class that we can call from and work with our old code with add-ons.

This project puts the team on a learning curve, after the understanding of the first assignment, the team is consistently looking for new ways to solve a problem. The game was scraped from the first assignment and adding methods that satisfies it needs, as the game was built from scratch during our first assignment. The team was able to gain experience working together, completing a project.

**Changes made from previous assignment**

From assignment 3 to assignment 2 to assignment 1, to further complete the game as a whole, we made a new design documentation file. Using the template of our previous assignment and the feedback from the marks to reflect on what needs to be done on the new design documentation.

This assignment’s documentation file uses the backbone of the first assignment such as the game board, which affects the behavior table. Using the foundation code, we merely added more methods and classes to satisfy the task given.

Changes are made to base method when they are affected, and by adding a new class for the AI component to work with the back bone of the assignment. Using a simple algorithm allowing the bot to play the game.

**Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Cases | Method of Testing | Expected results | Graphic Illustration/ Test Results |
| The game should include a board for 6 Man's Morris and 2 different colour disks | Start the game | The board to have 6 Man’s Morris and 2 different coloured disk | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.00.43).pngPassed |
| The user should be able to start a new game | This can be tested by clicking on New Game button | When the new game button is clicked, the player is randomly picked | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.02.45).pngPassed |
| The game should not accept a state where one piece is put on top of another | This can be checked by putting the pieces on the board and clicking the start button. A message will appear on the top for indicating if the state is valid or not. | Error will be displayed to be invalid move | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.03.04).pngFailed |
| The game should not accept a state where more than six piece of one colour are placed on the board | By trying to put more pieces on the board before starting | The Game should be in progressed to move the pieces | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.03.38).pngPassed |
| The game should not accept a state where there are no pieces on the board | By pressing start after new game | Error will be displayed to be invalid move | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.14.42).pngFailed |
| The game is able to recognize a Mill | By placing your piece in a form of a mill | The game shows you the state in progress of a Mill | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.04.42).pngPassed |
| After the game recognizes a mill, user is then able to remove other set piece | When game in progress states Mill, Click on any opposing color chips to remove them | The other player reduces their score by one and it is the next player’s turn to move | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.04.51).pngPassed |
| The game is able to recognize a winner | When either players has less than 3 chips, the player loses. | The game shows the state of a player’s win | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.06.59).pngPassed |
| The game is able to save | While having a few chips on the board and click on save button | The game shows the state of a game saved | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.05.46).pngFailed |
| The game is able to continue from previous save file | After a fresh new game, previous save file is loaded after pressing the load button (from previous test case) | By hitting load button at the beginning of the game | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.00.43).png  TO  C:\Users\yiker\Desktop\ss (2016-04-08 at 06.05.46).pngPassed |
| The game is able to play against the bot | By clicking on the Vs. Bot button to start playing against a bot | When the button is pressed the game proceeds to randomize which side to start first, player or bot. If the bot starts first, the side will place a chip on of the slot | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.15.42).png  Passed |
| The bot is able to remove a piece when a mill occurs for the bot | When the bot has a mill, will proceed to remove a piece from the player | When the bot has a mill, the number of player’s chip reduced by one | C:\Users\yiker\Desktop\ss (2016-04-08 at 06.17.32).png  Passed |