**Decompostion**

The checkers game is composed of five classes. It has been setup so that each class can be clearly categorized under each of the MVC sections. The *Mainframe* class and falls under the Model section while *Checkpiece, GameStatus* and *CheckBoardBlock* fall under the View section. The *menuitems* class handles inputs from users meaning it falls under Controller.

*Mainframe:*

The mainframe class models the checkers game using arrays. Arrays are used to represent physical objects of the game such as the checkers board and the pieces of the board. The checkers board was simulated with the use of a 2D array. The checkers pieces were represented by two separate, single dimension arrays. The mainframe is the class that handles the logic for checkers. It is responsible for simulating the the checkers game in a console version

Checkpiece:

The Checkpiece class handles the graphics for creating the individual pieces. This class graphically represents the state of the individual pieces. The logic from the mainframe class is fed to the checkpiece class, which in turn displays the state of the piece on the monitor

CheckBoardBlock:

Similar to the Checkpiece class, it too handles graphics. This class draws individual squares of the checkers board. Each individual block represent a single location in the 2D array simulating the checkers board.

GameStatus:

The GameStatus class handles the status of a checkers game

Menuitems:

This class is responsible for providing menuitems that GUI based (i.e buttons, drop-down menus, etc)

**MIS 1**

**Module: MenuItems.java**

Constants: public static final int itemsCount [], public static final string items []

Constructor: Static

* Initializes menu-items

**MIS 2**

**Module: Mainframe.java**

**Public class MainFrame implements ActionListener and JPanel**

Constructor Mainframe:

* Initializes menu-items

Public void initPieces()

* Initialize both white and block pieces

Public void Initialize()

* Board initialized and pieces on initial positions

Public void initCheckBoardAutomaticly()

* Initialize all pieces to default opening positions

Public void setBoardEnabled(boolean flag)

* make board clickable

public void initCheckBoardManually()

* let user define positions for pieces manually

public void putPieceOnBoard(CheckBoardBlock b, String player)

* triggered during manual setup, this puts one piece in the board

public void reFormatPieceLayer()

* Refresh the pieces according to the piece record

public int getBlackCountInVector()

* get black piece on the board

public int getWhiteCountInVector()

* get white piece on the board

public void print()

* prints checker piece for testing

public void actionPerformed(ActionEvent e)

* detects a clickable event and performs the appropriate operation

publc void mouseDragged(MouseEvent e)

* changes the position of the piece according to the mouse drag direction and destination

public CheckBoardBlock findMovedBlock(int x, int y)

* Returns the block on the board of the piece that was just moved

public void changePosition(CheckPiece p1, String newP)

* Move the a piece to a new location on the board

public void clearCapturedPiece(CheckPiece p1, String newP)

* Clears the piece that was captured

public void disablePlayer(String player)

* disables a player from moving

public void enablePlayer(String player)

* enables a player to move

public boolean isPeace()

* Checks to see if the game is at Peace

public String win()

* Checks to see if the game is over and returns the winner of the game

public void startGame()

* Starts the game

**MIS 3**

**Module: CheckBoardBlock**

**Types: Strings, Graphics**

public CheckBoardBlock(String color, String position)

* position and colour values are initialized by this constructor

public void paint(Graphics g)

* paints each block either red or white

**MIS 4**

**Module: CheckPiece**

public CheckPiece(String player, String position)

* Position and colour values are initialized

public void paintComponent(Graphics g)

* Paints the individual pieces

public void paintBorder(Graphics g)

* Paints the border

**MIS 5**

**Module: GameStatus**

**Types: Strings**

public String getStatus()

* returns the status of the game, whatever the may be

public void setStatusInit()

* sets the status to initialize

public void setStatusExec()

* sets the status to execute

public void setStatusOver()

* sets the status to over

public GameStatus()

* sets the game status to initialize

**Public Interface**

Menu

The object menuBar in the MainFrame module acts as the menu bar of the game. It uses the java package Menubar to implement it. The menu bar is populated with the items in the class MenuItems. The method actionPerformed detects which menu item has been clicked, and will perform the corresponding action.

Set Pieces

Automatically:

The pieces will be set automatically when the menu item "Initial Checkboard automatically" is pressed. This process is done by the initChesckBoardAutomaticly method of the MainFrame. This method calls on the methods initPieces and clearBoard. initPieces will reset the locations of the pieces are stored. The clearBoard method removes all the current pieces from the board. This is in case there is already a game in progress when the player decides to create a new game.

Manually:

The pieces will be set manually when the menu item "Initial Checkboard manually" is pressed. The initCheckBoardManually method is called, and once again, this method will call clearBoard to remove any pieces that were on the board before the menu button press. The state is changed to edit. setBoardEnabled will run and allow the board to be clickable, so the positions can be set. A button labeled “Switch Player” can be pressed, and will change the colour of the pieces being set. The mousePressed, mouseDragged, and mouseReleased method will change the location of a piece when it is dragged using the mouse to a new valid location on the board. A left click on a checker piece already placed on the board will toggle it to a king or back to a regular piece. When the “confirm button is pressed, the confirm variable will tell the game to start, and the state is set to play game. Hitting the confirm button will disable switcher, setBoardEnabled, and confirm.

Movement :

A piece can be moved when the game is the the proper state, and only a valid piece can be moved. The mousePressed method in the module MainFrame is in charge of moving pieces. Once a valid piece is pressed and held on to, the getMovablePosition method will run, and will find valid spots for target piece to move according to checkers rules. This method will also check if a piece has reached the end of their opponent’s side. If this is true, the piece will be assigned the king status, and will be subjected to different rules when moved, according to checkers rules. The letter “k” will be drawn on pieces as an indicator. When a value piece is clicked, the target will be drawn with a blue outline, and the valid move blocks will be outlined green. Methods mouseDragged and mouseReleased will then record the location of the mouse after a piece is dragged and released, for this location should be the location the player wants the piece to move. If the move is valid, the exchangeActivePlayer method will switch to the other player’s turn. The positions of the pieces will update.

Wins

During gameplay, every time the mouse is released and a piece is moved, the isPeace method will check if there is a winner in the game. The game will also check if there is a winner after the board has been manually set up, for the players could fail to put any piece of a specific colour on the board, which will result in an automatic win for one player. isPeace will modify the win and winner variables if there are no pieces of a single colour left. The game will then print “Winner is” followed by the winning colour.

Saving

Once the “save” button in the menu is pressed, the game will write a text file in the directory named save.txt. It will contain the current state of the game, and which player’s turn is active. The positions and status of the pieces will also be recorded.

Loading

Once the “load” button in the menu is pressed, the game will try and read a file in the directory named save.txt. It will modify the array containing the position of the pieces accordingly. It will then set up the board for playing, such as enabling all the actionlisteners, setting the states.

**Users relationship**

The uses relationship is basically viewed as hierarchical structure that explore the relationships between modules. It represents the dependencies modules have on each other and define how the services are implemented through interfaces. As mentioned, “uses” should be a hierarchy that makes software easier to understand, build and test. And each level defines an abstract machine for the next level, known as “level of abstraction”. For instance, let S be a set of modules, S = { M1, M2,.....Mn}, if Mi and Mj are in S, k is the maximum level of all nodes of Mj such that Mi is relational to Mj, then there must exists level k+1 in Mi.

In the second version of Checkers source code provided, “MainFrame”, “GameStatus”, “CheckPiece”, “CheckBoardBlock”, “DragButton”, “MenuItems”, “Test” modules. All of these modules are implemented in “MainFrame”.

“Jframe” created a top level container for the rest of an interfaces’s graphical component that provided a way to create a window for a java application. “JLayeredPane 1p” which is a “Jframe” layer object has split into two layers, top layer “Jpanel boardBg” which used to represent to checker game’s board and bottom layer “Jpanel body” used to describe checkers. Some examples of the description of the methods under “MainFrame” module are as follow.

MainFrame() generated menu bar and menu items, and set listener.

InitPieces() initialize the checker pieces collection (black and white ).

Initialize() In this method, initialize the check board, all pieces are on initialized position. It set up the layer object onto “Jpanel boardBg” by using “1p =this.getLayeredPane;”. Define the board by Jpanel with a 8 by 8 GridLayout.

clearBoard() Remove all pieces from board and clear template data.

setBoardEnabled() This set the mouse click action clickable or not.

initcheckBoardManually() ,initCheckBoardAutomaticly() These two method allow user to set the checker pieces manually on the board or automatically at the initialized position.

putPieceOnBoard() put the pieces on the board when user select manually generated pieces on board.

The “MainFrame” module works on the skeleton of the game, it provides the game structure, initialize board, initialize checkers, menu bar, determine initialize type, put pieces on the board, make movement of pieces, determine the validity of the moves, display result and whether restart the game when a game is over. The “CheckPiece” module works on the specifications of the checker pieces (eg. Color). The “GameStatus” record the status of the game and update the game. The “CheckBoard” mainly works on the graphical color of the board. “DragButton” provides the response that the program should do when the mouse make actions. And “MenuItems” initialize menu items.

The complex system is divided into modules that each module are highly cohesive because each components in the module are closely related to one another and modules have very low interactions with others which means low coupling.

**Private Implementation**

Module: GameStatus

It contains the variable “status”. This records the state of the game. When status is set as:

GAME\_STATUS\_INIT: the game has just been opened, and the game is in the initial state.

GAME\_STATUS\_EDIT: the game is allowing the players to manually set up the location of the checker pieces

GAME\_STATUS\_EXEC: the players can now play the game of checkers.

GAME\_STATUS\_OVER: the game has ended. This state is similar to the initial state.

Module: MainFrame

Here is the list of hidden objects in the main module.

Jpanel boardBg: Background layer of the checkerboard

JLayeredPane lp: frame layering of objects

JPanel body: layer of the checker pieces

GridLayout gridLayout: Layout of the checkerboard blocks

CheckPiece blackPieces: object containing the locations of black pieces, and will draw them as well

CheckPiece whitePieces: object containing the locations of white pieces, and will draw them as well

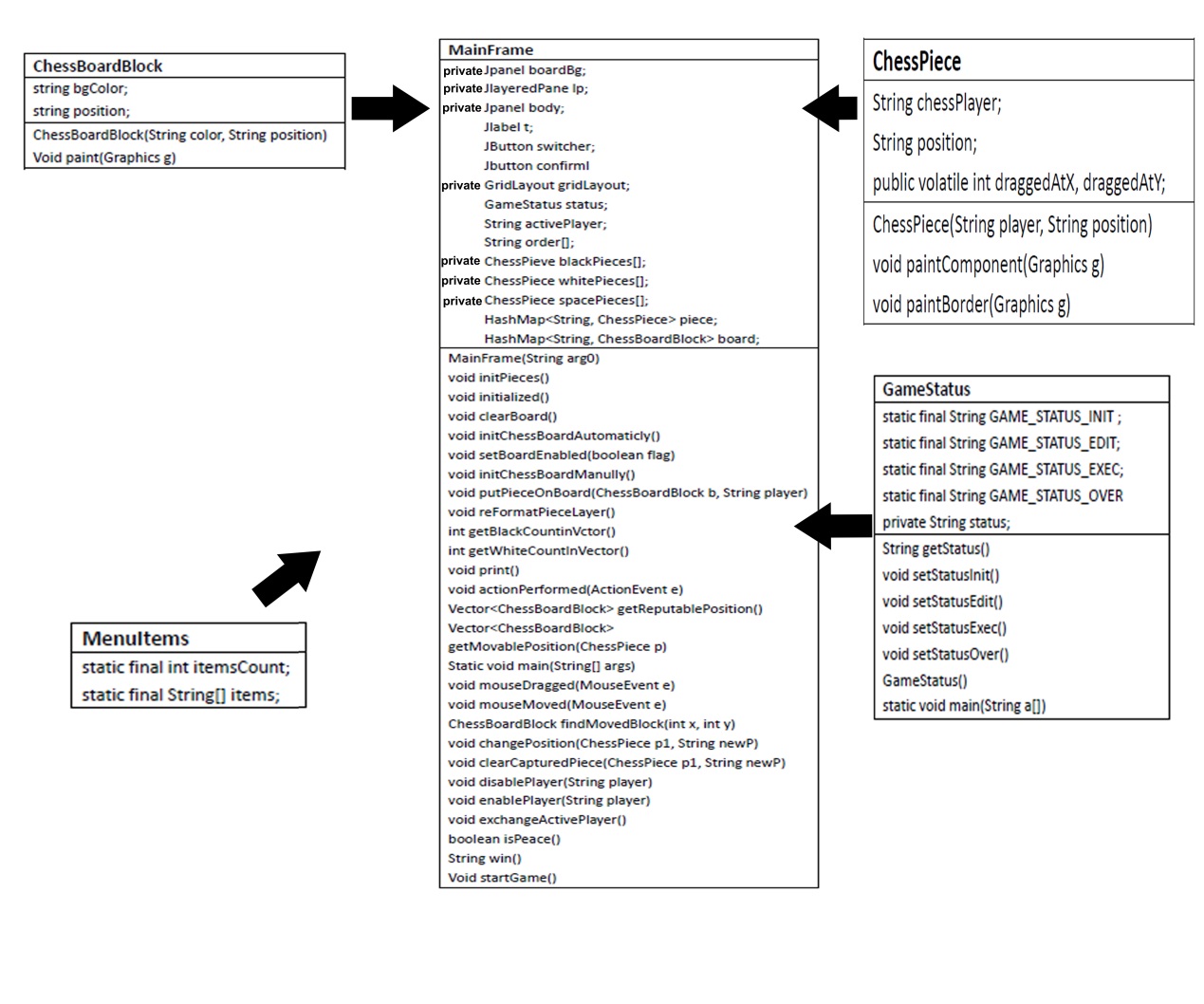
CheckPiece spacePieces: object containing the locations of the empty valid spaces on the board

**Traceability**

|  |  |
| --- | --- |
| Users are able to start a game from default starting positions | Class: MenuItem  Activates when user clicks the button Initial CheckBoard automatically from the menu  Class: MainFrame  Calls two methods:  initPieces()  Resets the location of the pieces  initcheckBoardAutomaticly() clears the board and put pieces on board according to default check starting rules |
| Users are able to set the starting positions of the pieces manually | Class: MenuItem  Activates when user clicks Initial checkBoard manually from the menu  Class: MainFrame  initcheckBoardManually()  Changes the status of the game to GAME\_STATUS\_EDIT, which enables mouse interaction to place the pieces. (left click to place piece, right click to set/cancel king)  Activate and set visible the button “switcher” to switch the player  Dragging an existing piece with the mouse will change its position to a valid location upon the mouse release  Activate and set visible the button “confirm” which confirms the placement of the pieces |
| Users are able to save one game | Class: MainFrame  saveGame()  creates the file (save.txt) then store the current player turn, positions of all the pieces and the game status |
| Users are able to load the saved game | Class: MainFrame  loadGame()  load the current player, position of all the pieces and the game status |
| Make moves | Class: MainFrame  changePosition(CheckPiece p1, String newP) – move piece(pi) to new position(newP)  clearCapturedPiece(CheckPiece p1, String newP) – Clear the piece if was captured  disablePlayer(String Player)/ enablePlayer(String player) – check if player can/cannot make the move  exchangeActivePlayer() – switch players  more comments in the code |

**Evaluation of Design :**

**5 = Outstanding 4 = Good 3 = Satisfactory 2 = Poor 1 = Unsatisfactory**



|  |  |
| --- | --- |
| An 8-by-8 checkers board with light and brown squares, with the light square at the bottom right  corner. | 5 |
| The user is able to set the initial position of pieces on the board at opening positions by menuitems. | 5 |
| Users are be notifyed if the position is illegal by green squares when drag piece, and pieces can only be placed on the highlighted squares. | 5 |
| Maximum numbers of white and black pieces are both 12. | 5 |
| All the components are shown in a graphical interface. | 4 |

**Testing:**

**Testing and Maintaining**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case: |  | Expected Results | Actual Results |
| 1) | Click to set a checker on already exists position | No response | No response |
| 2) | Click the very right corner of chessboard | No response | No response |
| 3) | Click a valid position | The application should set piece on the board. | The application should set piece on the board. |