Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

Documentation

Bookmarks: Requirements Input/output UI Guide 4.1 4.2 4.3 4.4 4.5 4.6 4.7 Test Report

Requirements

1. Game start with a Menu display
2. Menu has two buttons, New Game and Custom Game
3. New Game switches to a blank display (implemented in assignment 2) with a button to go back to the Main Menu
4. Custom game switches to the game display which contains the board, buttons to switch between discs, reset the board, check the state, go back to the Main Menu, Labels to display the error messages and to display the activated disc
    a. Initially a random disc is activated
    b. Clicking on the Red Disc button activates the red disc and clicking on the Blue Disc button activates the blue disc
    c. Clicking in a slot of the board puts the activate colored disc into that slot
    d. Clicking on the same slot twice with the same color removes the disc from that slot
    e. Clicking on the same slot twice with a different color replaces the slot with the different color
    f. Clicking on the Reset button makes all the slots empty
    g. Clicking on the Check State button shows if there are or there aren't any errors with the current configuration on the board and shows them inside a Label
    h. Possible errors include:
        i. Too many of one colored discs (i.e. color A > color B + 1)
        ii. Floating disks (i.e. discs with empty slot underneath them)
        iii. Four discs of the same color are together (horizontally, vertically or diagonally)
    i. Clicking on the Main Menu button switches the display back to the main menu (and resets everything inside Custom Game display)

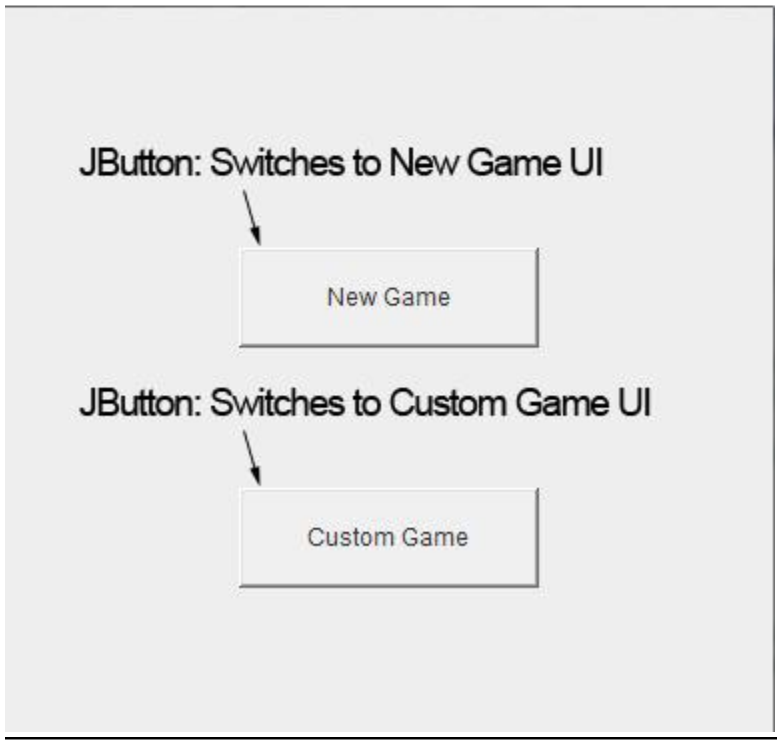
Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

Input/output

Inputs		Outputs	
- Click red disc button	M_red	- Red disc active	c_redActive
- Click blue disc button	M_blue	- Blue disc active	c_blueActive
- Click empty slot	M_slotEmpty	- Color slot red	c_slotRed
- Click occupied slot with red disc	M_slotOccupiedRed	- Color slot blue	c_slotBlue
- Click occupied slot with blue disc	M_slotOccupiedBlue	- Make slot empty	c_slotEmpty
- Click check state button	M_check	- Board state is valid	c_valid
- Click reset board button	M_reset	- Board state is invalid	c_invalid
- Click main menu button	M_mainMenu	- Empty board	c_empty

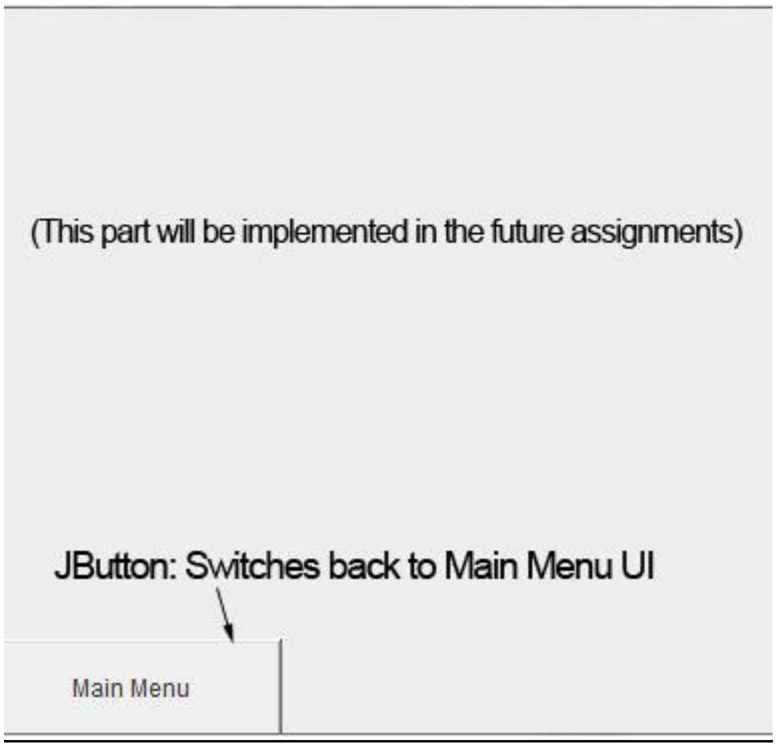
User Interface Guide

Main Menu:

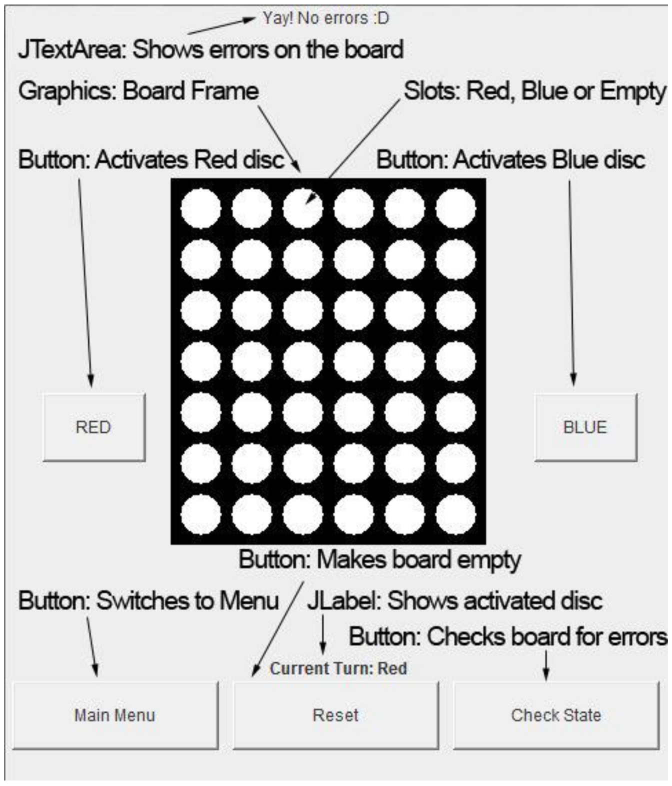

JButton: Switches to New Game UI
New Game
JButton: Switches to Custom Game UI
Custom Game

Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

New Game:


(This part will be implemented in the future assignments)
JButton: Switches back to Main Menu UI
Main Menu

Custom Game:


Yay! No errors :D
JTextArea: Shows errors on the board
Graphics: Board Frame
Slots: Red, Blue or Empty
Button: Activates Red disc
Button: Activates Blue disc
RED
BLUE
Button: Makes board empty
Button: Switches to Menu
JLabel: Shows activated disc
Button: Checks board for errors
Current Turn: Red
Main Menu
Reset
Check State

4.1

We decided to split our project into 4 classes: ConnectFourController, ConnectFourModel, ConnectFourView and ConnectFourMain.

1. The ConnectFourController class receives, interprets and validates input from the user and updates the Model and View classes.
2. The ConnectFourModel class stores the current configuration of the board and it does the calculations for checking the configuration. It's accessed by the controller class to check and update the configuration.
3. The ConnectFourView class draws the UI for the Connect Four game.
4. The ConnectFourMain class creates the objects for the Model, View and Controller.

MVC allows to separate business logic from UI logic in the program. It allows for loose coupling, thus more specific tasks can be assigned to the members in the group. Adding on to that, testing of the modules also becomes a lot easier and more efficient. Although this application is quite simple right now, as we add complexity, we can use the MVC rules to easily breakdown where each new component falls under.
To be more specific, our controller class handles the communication b/w the model and the view. It gets the input from the user, notifies the model about it, gets the updated model and updates the view accordingly. The model handles and stores the configuration of the board. It also does the calculations to check if the current configuration would result in an error. Finally, the view is responsible to draw the buttons, labels and the board (with all the slots). It redraws itself every time the controller sends a new updated configuration of the board. Finally, the main class initializes the model, view and controller and makes the initial view visible.

4.2

ConnectFourController class contains the following public methods:

actionPerformed (Action Event e**):**

1. Checks which button is pressed (Main Menu, Custom Game, New game, Reset, Red, Blue, Check State).
2. This is done by using if statements along with .getSourceLabel() and .equals() that correspond to each possible button press. Note: it updates the View if the Red or Blue buttons are pressed so the client knows whose turn it currently is via on screen text.
3. After checking which button has been pressed, it changes the Game State in the model accordingly using the Enum GameState in the Model class.
4. It then updates the View to reflect the new Game State by calling the switchScreen method in the View class.

mouseClicked():

1. Checks for mouse clicks during the game session (if the current state is the Main Menu, it returns null).
2. Calculates the tile position that corresponds to the mouse click. This is done by getting the board coordinates from the getBoardCoordinateofPoint methodin the View class.
3. Checks to insure the board coordinates (both x and y) are in range of the board by using the getBoardConfiguration method from the Model class and ensuring X and Y are not less than 0 and are not greater than the length and columns of the board array and stops the function if it is.

<u>**ConnectFourModel** class contains the following public methods</u>:

checkBoardConfiguration (Slot[][] newBoardConfiguration**):**

- This method takes in an array with the new board configuration to check if it's valid or not

1. Checks if the amount of blue discs and red discs are valid using the getRedDiscCount and getBlueDiscCount private methods with the newBoardConfiguration.
2. If not display "Too many blue discs" or "Too many red discs" depending on the error on custom game window.
3. Checks for floating discs by using a series of for loops and if statements.
4. If floating disc(s) is found display "Floating disc(s) detected" on custome game window.
5. Check for 4 discs in a row of the same colour using the private method.
6. If 4 discs are in a row display "Error four discs of the same colour together"

getWinState(Slot[][] boardConfiguration**):**

- This method checks whether or not the win state (4 red or blue in row) has been reached

1. Checks four possible win states for red and blue: Vertical, Horizontal, Diagonally Right to Left (looking down), Diagonally Left to Right (looking down).
2. Compares to 3 discs beside the current disc in the loop in every direction and check if all are the same colour.
3. If this is true for any of the win conditions/directions, return true, otherwise false.

*NOTE: the following methods are used by the controller to obtain or set the instance variables.

getTurn()
- This method simply returns the currently active state of colour (Red or Blue) using the currentTurn instance variable.

setTurn(slot newTurn**)**

- This method sets the new active state of colour for currentTurn if the current state is not an empty slot. I

getErrorMessage ()
- Returns the string with the correct error message using the instance variable errorMessage.

getBoardConfiguration()
- Returns the boardConfiguration instance variable that represents the board in a 2D array.

setGameState(GameState g**)**
- Sets the new GameState.

getGameState()
- returns the current GameState

resetConfiguration()
- This method goes through evey slot in the array and sets it to empty using a nested for loop and setting the instance variable Slot to the empty state.

ConnectFourView class contains the following public methods:

ConnectFourView(ConnectFourModel.Slot[][] b, ConnectFourModel.GameState gs**)**
- This is the constructor that sets up the board of size 400 by 400 pixels using the 2D array b and gamestate gs, makes the window closable and sets the size of the screen (400 by 400).

ConnectFourView(int width, int height, ConnectFourModel.Slot[][] b, ConnectFourModel.GameState gs**)**
- This constructor is exactly the same as the previous one except it lets you set the size to whatever you wish.

setTurn(ConnectFourModel.Slot turn**)**
- Gets the colour state of the board (turn) and displays the current turn on screen. IE “Current Turn: Red”.

setError(String error**)**
- Gets the string error message, sets it and displays it on screen.

addCalculateListener(ActionListener listenForButton, MouseInputListener listenForMouseEvents**)**
- Adds all the listener for the buttons. It gets the ActionListenr and MouseInputLister from the controller class.

displayErrorMessage(String errorMessage**)**
- This will be used later in the second assignment to display a popup error.

paint(Graphics g**)**

- This method redraws the board depending on the current GameState.

adjustBoard(ConnectFourModel.Slot[][])

- This method can be used to update the board, might become useful in later assignments.

switchScreen(ConnectFourModel.GameState currentGameState**)**

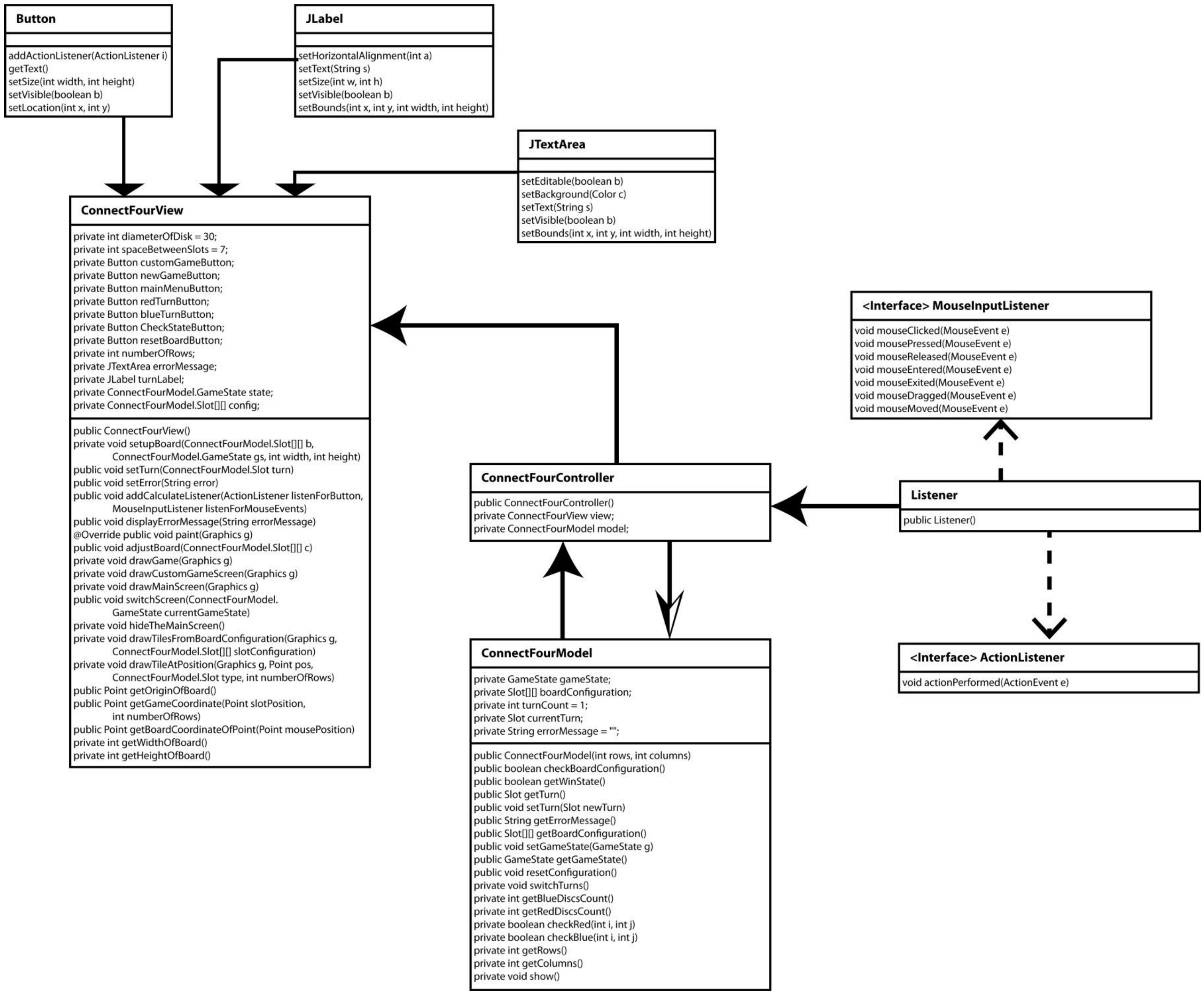
- This method has two purposes, it switches the screen based on the currentGameState and also checks if the currentGamestate is Main Menu. If it is, it removes any potential error messages that were being displayed and reverts back to the Main Menu screen.

getGameCoordinate(Point slotPosition, int numberOfRows**)**

- Converts the position of the game board with respect to the position of the array. It does this by taking the slot position inside the 2D array and converting it to the correct position on the screen.

Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

4.3 – Class Relation Diagram


Button
addActionListener(ActionListener i)
getText()
setSize(int width, int height)
setVisible(boolean b)
setLocation(int x, int y)
JLabel
setHorizontalAlignment(int a)
setText(String s)
setSize(int w, int h)
setVisible(boolean b)
setBounds(int x, int y, int width, int height)
JTextArea
setEditable(boolean b)
setBackground(Color c)
setText(String s)
setVisible(boolean b)
setBounds(int x, int y, int width, int height)
ConnectFourView
private int diameterOfDisk = 30;
private int spaceBetweenSlots = 7;
private Button customGameButton;
private Button newGameButton;
private Button mainMenuButton;
private Button redTurnButton;
private Button blueTurnButton;
private Button CheckStateButton;
private Button resetBoardButton;
private int numberOfRows;
private JTextArea errorMessage;
private JLabel turnLabel;
private ConnectFourModel.GameState state;
private ConnectFourModel.Slot[][] config;
public ConnectFourView()
private void setupBoard(ConnectFourModel.Slot[][] b, ConnectFourModel.GameState gs, int width, int height)
public void setTurn(ConnectFourModel.Slot turn)
public void setError(String error)
public void addCalculateListener(ActionListener listenForButton, MouseInputListener listenForMouseEvents)
public void displayErrorMessage(String errorMessage)
@Override public void paint(Graphics g)
public void adjustBoard(ConnectFourModel.Slot[][] c)
private void drawGame(Graphics g)
private void drawCustomGameScreen(Graphics g)
private void drawMainScreen(Graphics g)
public void switchScreen(ConnectFourModel. GameState currentGameState)
private void hideTheMainScreen()
private void drawTilesFromBoardConfiguration(Graphics g, ConnectFourModel.Slot[][] slotConfiguration)
private void drawTileAtPosition(Graphics g, Point pos, ConnectFourModel.Slot type, int numberOfRows)
public Point getOriginOfBoard()
public Point getGameCoordinate(Point slotPosition, int numberOfRows)
public Point getBoardCoordinateOfPoint(Point mousePosition)
private int getWidthOfBoard()
private int getHeightOfBoard()
<Interface> MouseInputListener
void mouseClicked(MouseEvent e)
void mousePressed(MouseEvent e)
void mouseReleased(MouseEvent e)
void mouseEntered(MouseEvent e)
void mouseExited(MouseEvent e)
void mouseDragged(MouseEvent e)
void mouseMoved(MouseEvent e)
ConnectFourController
public ConnectFourController()
private ConnectFourView view;
private ConnectFourModel model;
Listener
public Listener()
<Interface> ActionListener
void actionPerformed(ActionEvent e)
ConnectFourModel
private GameState gameState;
private Slot[][] boardConfiguration;
private int turnCount = 1;
private Slot currentTurn;
private String errorMessage = "";
public ConnectFourModel(int rows, int columns)
public boolean checkBoardConfiguration()
public boolean getWinState()
public Slot getTurn()
public void setTurn(Slot newTurn)
public String getErrorMessage()
public Slot[][] getBoardConfiguration()
public void setGameState(GameState g)
public GameState getGameState()
public void resetConfiguration()
private void switchTurns()
private int getBlueDiscsCount()
private int getRedDiscsCount()
private boolean checkRed(int i, int j)
private boolean checkBlue(int i, int j)
private int getRows()
private int getColumns()
private void show()

4.4

ConnectFourView:

- When game is run, the view class creates a window with "New Game" and "Custom Game" buttons.
- New Game (Part of Assignment 2 and Assignment 3) has a Main Menu button displayed.
- In Custom Game mode, a board is set up with 6 columns and 7 rows.
- View class also sets up the buttons for:
    - Picking the colored discs.
    - Returning to Main Menu.
    - Checking state of the board (if the current state is valid).
    - Resetting the board (clears the board).
- Displays whose current turn it is as text(Red turn or Blue turn).
- Clicking on an empty spot on board draws the activated disc on that spot.
- If an already used spot on the board is clicked, the spot is erased if the activated disc is of the same color.
- If an already used spot on the board is clicked, the spot is updated with the new color if the activated disc is of a different color.
- If any of the errors checked in Model class exist, the view class displays a text regarding the error is displayed on the window.

ConnectFourController:

- When the Main Menu button is clicked in Custom Game mode or New Game mode, the user is redirected to the Main Menu.
- Custom Game redirects to Custom Game mode.
- New Game redirects to New game mode.
- Controller class checks any button that is pressed in the application.
- If a button is pressed, Controller class performs the action with respect to that button.
- When a board is clicked, Controller class checks if the position of the click is valid for a move.

ConnectFourModel:

- When in Custom Game mode, the first disc color is chosen randomly in the model class.
- Checks whose current turn it is (Red turn or Blue turn).
- When a disc (Blue or Red) is clicked, that color is activated.
- After setting up the board, the user checks state. The model class checks for any errors in the setup of the board;

- Checks if there are invalid number of discs (more than 1 difference between the two colored discs).
- Checks if there are more than 3 consecutive same colored discs.
- Checks if there are any floating discs (a disc with no disc underneath it).

4.5

ConnectFourView Private Entities:

VARIABLES

private int diameterOfDisk = 30;
- This variable holds the diameter of the disk
- Currently this variable does not get changed but it will need to get changed for assignment 2 because we will need to make the size of the board resize as the screen resizes

private int spaceBetweenSlots = 7;
- This variable holds the difference of the center of one disk with its horizontal
- Currently this variable does not get changed but it will need to get changed for assignment 2 because we will need to make the size of the board resize as the screen resizes

private Button customGameButton;
- this variable holds a reference to the button in the main menu that goes to the custom game
- this button is repositioned in the paint helper function, it is initialized in the constructor of the game
- the function hideTheMainScreen its visibility is set to false

private Button newGameButton;
- this holds a reference that to the button located in the main menu that goes to the game
- this button is repositioned in the paint helper function, it is initialized in the constructor of the game
- the function hideTheMainScreen its visibility is set to false

private Button mainMenuButton;
- this is the button in the both the game screen and the custom game screen that will take you back to the main screen

- this button is positioned in the paint helper function, resized and initialized by the constructor.

private Button redTurnButton;
- this holds a reference to a button that makes the turn the red players
- After pressing the button it will make the next disk that appears on the screen the color red

private Button blueTurnButton;
- this holds a reference to a button that makes the turn blue players
- After pressing the button it will make the next disk that appears on the screen the color blue
- this button is positioned in the paint helper function, resized and initialized by the constructor.
- set by the controller in the function setTurn

private Button CheckStateButton;
- this holds a reference to a button that tells the user the current errors in their board.
- this button is located in the custom game screen
- for example if the board had a floating piece or if the number of disks were not evenly colored, or if one of the colors already won

private Button resetBoardButton;
- this variable holds a reference to a button that tells the board to remove all disks from it
- the button is located in the custom game screen
- this button is positioned in the paint helper functions, resized and initialized by the constructor.

private int numberOfRows;
- This variable holds the number of rows of the board.
- it is changed in the drawBoard function and is used to draw the board and the tiles

private JTextArea errorMessage;
- this is a JTextArea that displays the error message
- It is located in the game screen at the top middle
- This variable is initialized and resized in the constructor, it is also repositioned in the paint helper function

private JLabel turnLabel; //displays the turn

- This variable shows as a string which players turn it is, meaning the next disc that is placed will be at that position
- This variable is initialized and resized in the constructor, it is repositioned by the paint helper function, and the labels text is updated by a setter that is called by the controller

private ConnectFourModel.GameState state;
- this variable holds the current state of the game for example MainMenu, Game, CustomGame
- It is used by the paint function to figure out how to draw the board, It is updated by a setter which is called from the controller

private ConnectFourModel.Slot[][] config; //holds the configuration of the game
- this holds the configuration of the board for each tile on the board, it is initialized by the constructor

private getOriginOfBoard()
- This method gets the origin of the board. Note the origin is the top left of the array/board and starts at slot 0.

FUNCTIONS

public ConnectFourView(ConnectFourModel.Slot[][] b, ConnectFourModel.GameState gs)
- This constructor is when the user does not give the size of the board so this creates a screen with the default screen size.

public ConnectFourView(int width, int height, ConnectFourModel.Slot[][] b, ConnectFourModel.GameState gs)
- this is a constructor, it creates the view object with the given arguments
- width = the horizontal distance of the screen
- height = the vertical distance of the screen
- b = this is a 2d array that represents the state of each slot on the board. The state being either red, blue or empty
- gs = the game state, for example mainMenu, CustomGame, Game

//PURPOSE: sets up the game screen
private void setupBoard(ConnectFourModel.Slot[][] b, ConnectFourModel.GameState gs, int width, int height)

- This function is called by the constructor and is used to setup the information such as creating all the buttons, resizing all the buttons, and adding them to the jframe
- width = the horizontal distance of the screen
- height = the vertical distance of the screen
- b = this is a 2d array that represents the state of each slot on the board. The state being either red, blue or empty
- gs = the game state, for example mainMenu, CustomGame, Game

//sets label depending on which color's turn it is
public void setTurn(ConnectFourModel.Slot turn)
  This function updates the turn label by setting its turn
    turn = the current turn such as blue or turn

public void setError(String error)
  - this function updates the error message in the error textfield
    error = a string to display on the screen representing the message

//PURPOSE: this function makes it the listener for the buttons the controller object
public void addCalculateListener(ActionListener listenForButton, MouseInputListener listenForMouseEvents)
  - this function sets the button and screen listener for their corresponding buttons
    listenForButton = the ActionListener that will respond to the button presses
    listenForMouseEvents = this is the mouse event listener that will respond to button presses even if they are not on the buttons

//PURPOSE: displays and message on the screen of the text errorMessage that will be passed in as a parameter
public void displayErrorMessage(String errorMessage)
  - this function takes the error message and displays it in as a popup menu
    errorMessage = The message to display in the popup

//Purpose: if it is in the GameState then it will draw the entire screen
public void paint(Graphics g)
  - This function draws the entire screen and updates all the buttons that need to be updated and resized
    g = the default Graphics that is used to draw on the screen

public void adjustBoard(ConnectFourModel.Slot[][] c)

private void drawGame(Graphics g)
 - this function is called by the paint function it draws the game screen
  g = is the graphics object that is used to draw on the screen

//PURPOSE: it draws the game screen
private void drawCustomGameScreen(Graphics g)
 - this function draws the custom game, it does this by hiding the main menu, repositioning all the buttons, drawing the conect four board
  g = is the graphics object that is used to draw on the screen

//PURPOSE: it draws the main menu
private void drawMainScreen(Graphics g)

//PURPOSE: hides the menu from he screen
private void hideTheMainScreen()

drawTileAtPosition(Graphics g, Point pos, ConnectFourModel.Slot type, int numberOfRows**)**
- The first argument is the graphics object used to help draw on the screen (IE. Setfill changes the colour for the next thing drawn). The second argument is just the point position of the circle. The third argument is the slot type (Red, Blue,Empty) and the fourth argument passes in the number of rows. This method draws one tile at the specified location.

drawTilesFromBoardConfiguration(Graphics g, ConnectFourModel.Slot[][] slotConfiguration**)**
- The first argument is used to help draw on the screen, the second is the 2D array configuration of the board. This calls the drawTileAtPosition in a nested loop so all the tiles can be drawn in sequence.

hideTheMainScreen()
- This method hides the menu buttons when in game.

drawMainScreen(Graphics g**)**
- This draws the menu screen and the argument passed in is the graphics object used to help draw it. It's other purpose is to hide all the menu buttons except the ones being used (New Game and Custom Game).

4.5 (Continued)…

ConnectFourController Private Entities:

Instance Variables:

Private ConnectFourView V**iew:** The View class object used to update the view and notify the controller.

ConnectFourModel Private Entities:
The Model class object used to update the model and notify the controller.

The ConnectFourController class contains no private methods.

ConnectFourModel Private Entities:

Instance Variables

private GameState gameState; Keeps track of the current state of the game (Game, Custome Game or Main Menu)
private Slot[][] boardConfiguration; 2D array to represent the board configuration
private int turnCount = 1; Used for switchTurn which will become handy in Assignment 2 or 3.
private Slot currentTurn; Represents the currently active colour state of the board.
private String errorMessage = ""; The currently set error message to be displayed.

Private Methods:

getBlueDiscsCount(Slot[][] boardConfiguration**)**
*****This method given the current board configuration 2D array, counts the amount of blue discs and returns the amount.

getRedDiscsCount(Slot[][] boardConfiguration**)**
*****The same functionality as the getBlueDiscCount method but for red discs.

checkRed(int i, int j)
*Passes in row and column number for a certain slot and checks if that slot is red.

checkBlue(int i, int j)
*Same functionality as checkRed but for blue discs

4.6

Pros	Cons
- By implementing the MVC method, the tasks presented in Assignment 2 and 3 should be easier - The entire MVC structure is maintained fairly well; all the separate modules perform their specific functionality - View and Model not interacting made individual work easier (changes would not affect the other class)	- Our module interaction could've be improved by further modularizing the view class - The UI could've been improved for better user experience - The use of the swing class is not as efficient as it can be; should've split the display into separate panels instead of drawing directly on to the JFrame object - Encapsulation could have been implemented better, we could have utilized more Classes for the View

4.7

All the documentation for the code is in the source code files provided in the submitted zip folder.

Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

<u>Test Report</u>

We tested our application by treating each of the requirements as a test case:

Game start with a Menu display	Check
Menu has two buttons, New Game and Custom Game	Check
New Game switches to a blank display (implemented in assignment 2) with a button to go back to the Main Menu	Check
Custom game switches to the game display which contains the board, buttons to switch between discs, reset the board, check the state, go back to the Main Menu, Labels to display the error messages and to display the activated disc	Check
Initially a random disc is activated	Check
Clicking on the Red Disc button activates the red disc and clicking on the Blue Disc button activates the blue disc	Check
Clicking in a slot of the board puts the activate colored disc into that slot	Check
Clicking on the same slot twice with the same color removes the disc from that slot	Check
Clicking on the same slot twice with a different color replaces the slot with the different color	Check
Clicking on the Reset button makes all the slots empty	Check
Clicking on the Check State button shows if there are or there aren't any errors with the current configuration on the board and shows them inside a Label	Check
Possible errors include: - Too many of one colored discs (i.e. color A > color B + 1) - Floating disks (i.e. discs with empty slot underneath them) - Four discs of the same color are together (horizontally, vertically or diagonally)	Check
Clicking on the Main Menu button switches the display back to the main menu (and resets everything inside Custom Game display)	Check