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 starts with a Menu display
2. Menu has four buttons, ***Player vs Player***(two player game mode), ***Custom Game, Player vs CPU*** and ***Load Save State***.
3. ***Player vs Player*** and ***Player vs CPU*** switches to a blank board where two players(or Player vs CPU) can start a game. There are three displayed buttons, ***Main Menu***(redirects to main menu), ***Reset***(resets the board) and a ***Save State***(saves the state of the game).
   a. The first turn(red or blue) is activated randomly.
   b. Clicking on any column drops the activated disc in that column.
   c. The colour switches after each turn.
   d. Clicking on the Reset button resets the board (all slots are empty).
   e. Clicking on the Save State button saves the state if the board is in a valid state.
   f. Blue Won or Red Won message is displayed depending on which player wins (When four discs of the same color are together horizontally, vertically or diagonally).
   g. Clicking on the Main Menu button switches the display back to the main menu (and resets everything inside Play game display.
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 Save State button saves the state if the board is in a valid state.
   j. 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

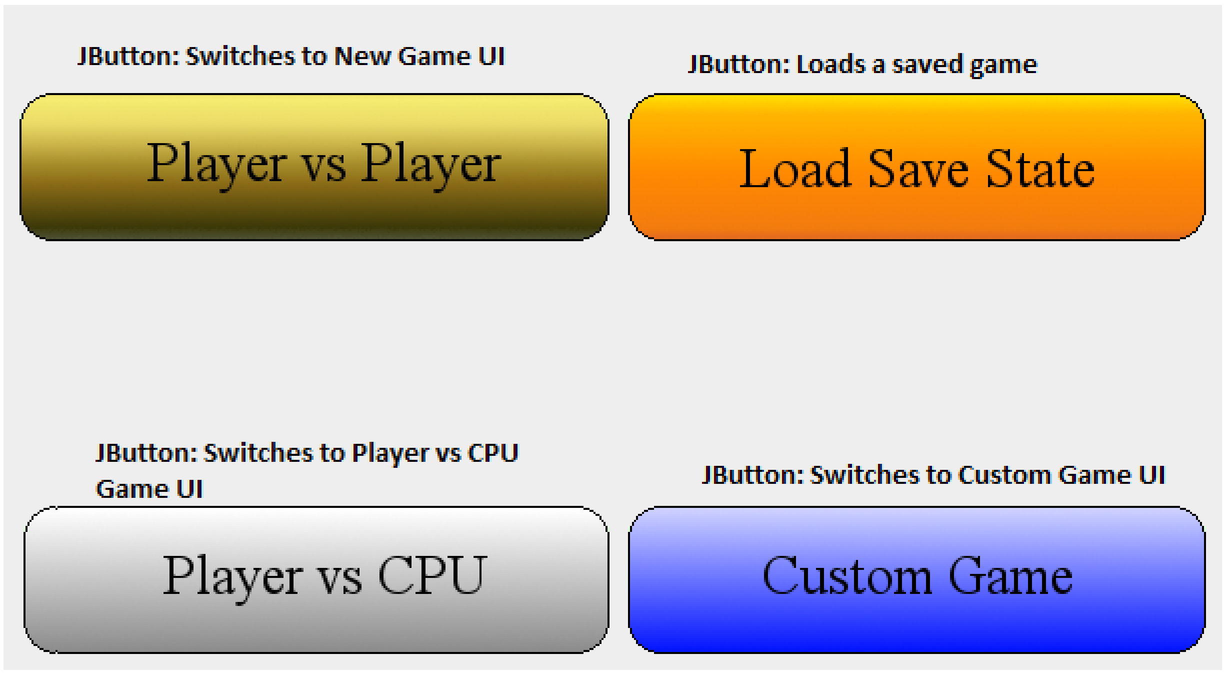
Input/output TOP

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

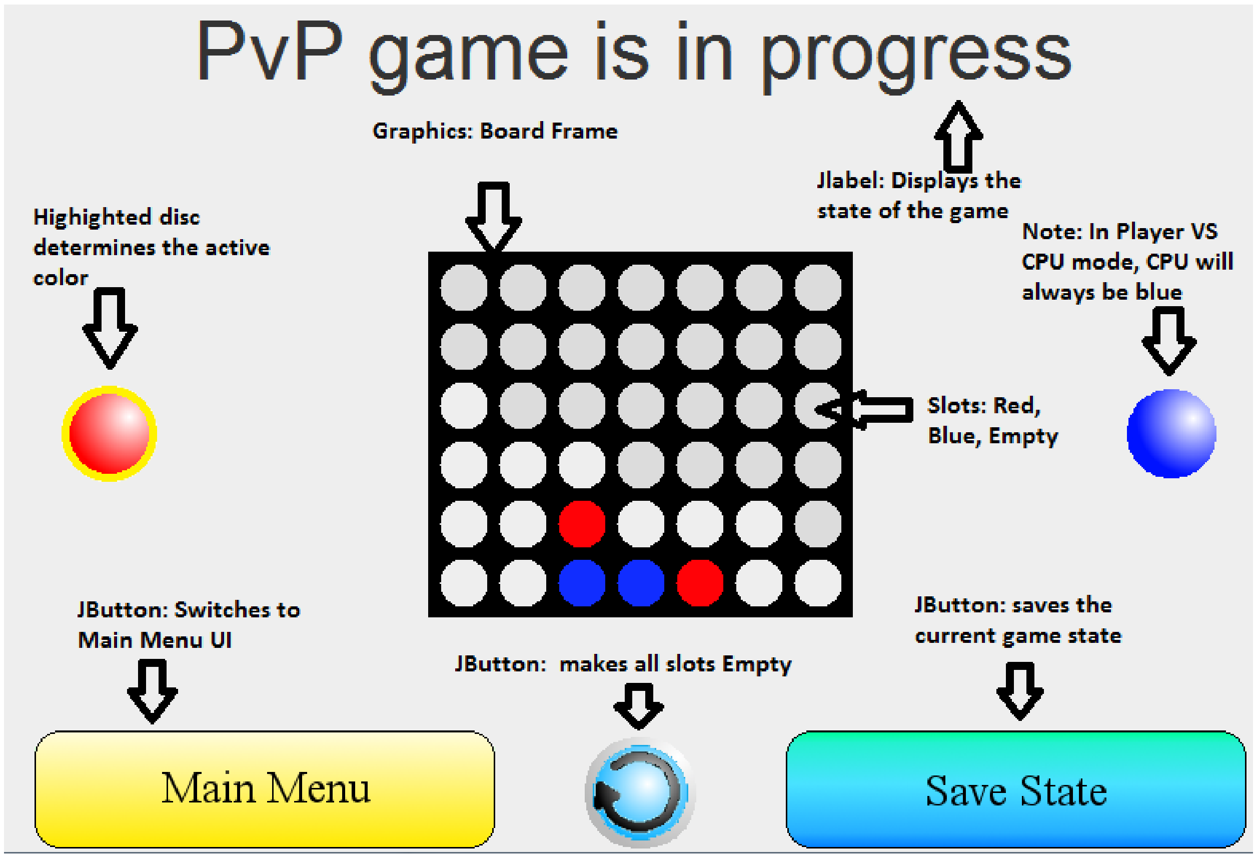
Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

User Interface Guide TOP

Main Menu:

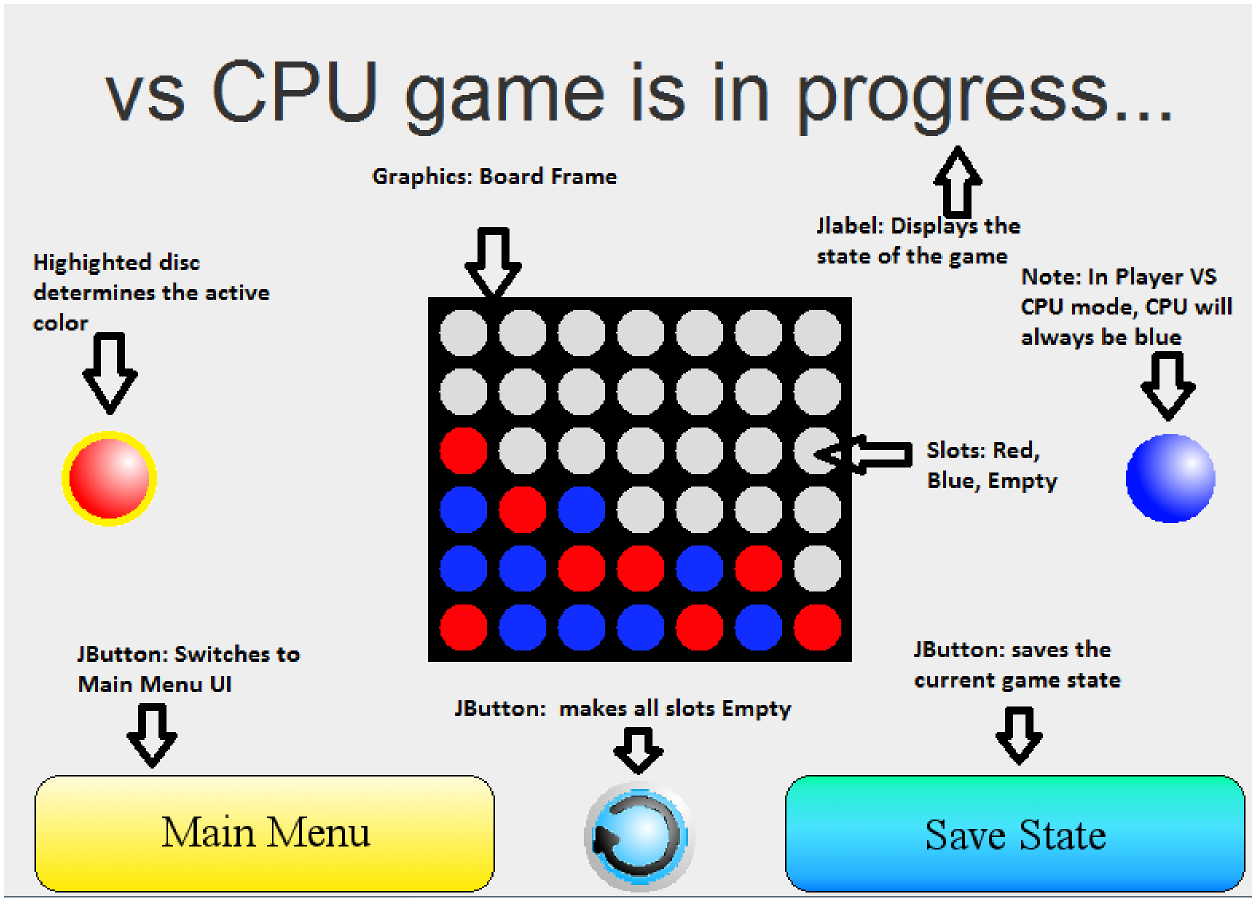

JButton: Switches to New Game UI
Player vs Player
JButton: Loads a saved game
Load Save State
JButton: Switches to Player vs CPU Game UI
Player vs CPU
JButton: Switches to Custom Game UI
Custom Game

Player vs Player:


PvP game is in progress
Graphics: Board Frame
Jlabel: Displays the state of the game
Highighted disc determines the active color
Note: In Player VS CPU mode, CPU will always be blue
Slots: Red, Blue, Empty
JButton: Switches to Main Menu UI
JButton: makes all slots Empty
JButton: saves the current game state
Main Menu
Save State

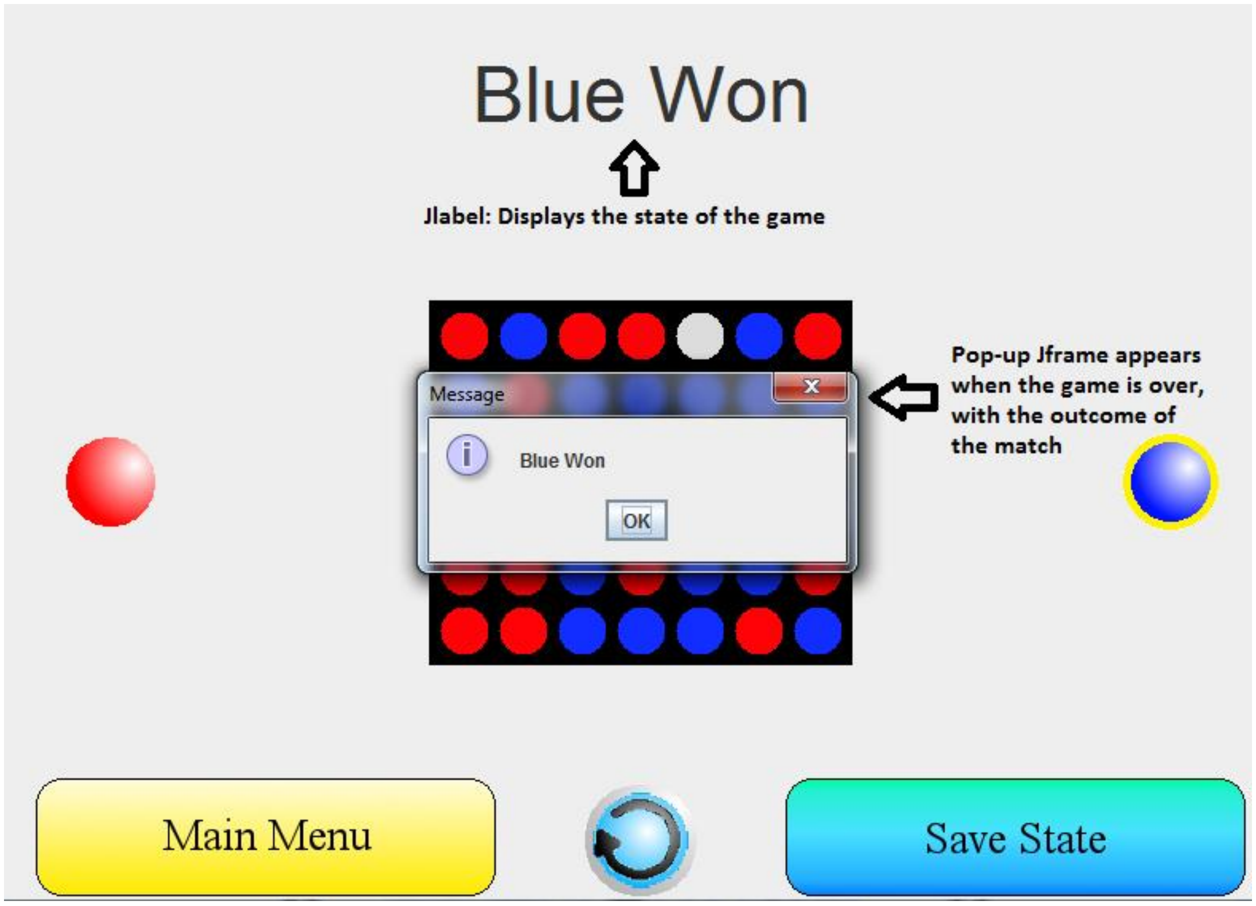
Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

Player vs CPU:


vs CPU game is in progress...
Graphics: Board Frame
Jlabel: Displays the state of the game
Highighted disc determines the active color
Note: In Player VS CPU mode, CPU will always be blue
Slots: Red, Blue, Empty
JButton: Switches to Main Menu UI
JButton: makes all slots Empty
JButton: saves the current game state
Main Menu
Save State

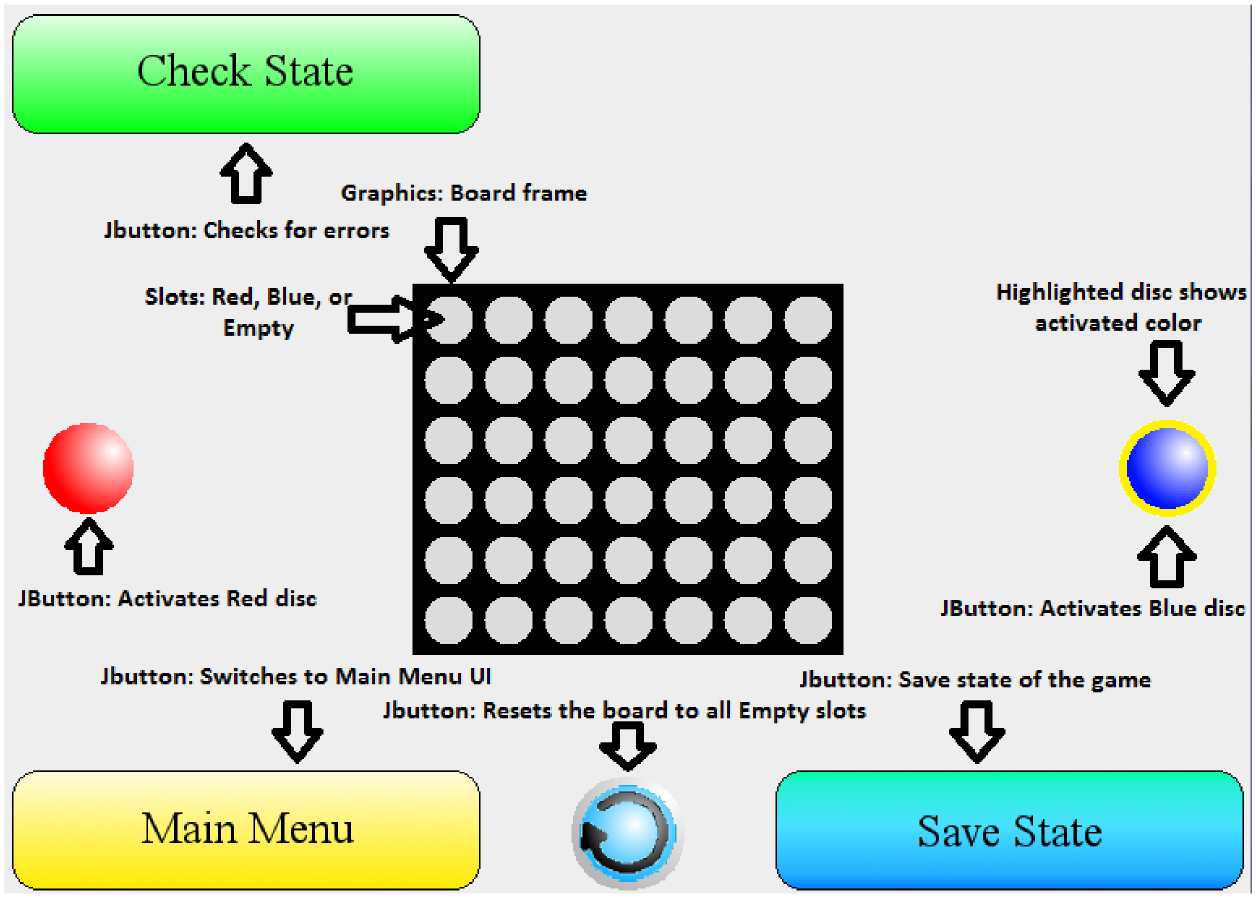
Created by Ahmed Khan, Saim Malik, Zayan Imtiaz, Aleem Ul Haq, Sergio Agraz

Player vs Player / Player vs CPU end state (no end state for Custom Game):


Blue Won
Jlabel: Displays the state of the game
Message
Blue Won
OK
Pop-up Jframe appears when the game is over, with the outcome of the match
Main Menu
Save State

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

Custom Game:


Check State
Jbutton: Checks for errors
Graphics: Board frame
Slots: Red, Blue, or Empty
Highlighted disc shows activated color
JButton: Activates Red disc
JButton: Activates Blue disc
Jbutton: Switches to Main Menu UI
Jbutton: Save state of the game
Jbutton: Resets the board to all Empty slots
Main Menu
Save State

4.1

TOP

We decided to split our project into 5 classes: ConnectFourController, ConnectFourModel, ConnectFourView, ConnectFourMain and Board.

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.
5. The Board class draws the board for the game.

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. The view is responsible to draw the buttons and labels. It redraws itself every time the controller sends a new updated configuration of the board. The board class draws the board including the animation for falling pieces (gravity). Finally, the main class initializes the model, view and controller and makes the initial view visible.

4.2 TOP

ConnectFourController class contains the following public methods:

actionPerformed(ActionEvent e**):**

1. Checks which button is pressed (Main Menu, Custom Game, Player vs Player, Player vs CPU, Load Save State, 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.

handleCustomGameState(MouseEvent e**):**

1. Handles all the mouse events that occur, such as clicking the board etc.
2. Get the mouse position of the click and then convert that to the board position where it will be in a different coordinate system.
3. The board coordinates are calculated using getBoardCoordinateOfPoint(mousePosition)

ConnectFourModel.Slot[][] getConfiguration():

1. Returns current configuration of the board.
2. This function is called by the view to get the configuration so that the view can draw it.
3. This function works by getting the board configuration from the model, model.getBoardConfiguration(), and then returns it.

showWinner():

1. Checks if a player has won by calling getGameProgess() and then check if (getGameProgess() == GameProgress.blueWon) or (getGameProgess() == GameProgress.redWon)
2. Displays win message, view.displayMessage("Blue Won") or view.displayMessage("Red Won").
3. Updates text on the screen to show the winner.
4. If theres a tie (getGameProgess() == (GameProgress.tieGame)) displays, “You both lose” message.

switchTurn():

1. Checks whose turn it was using getTurn()
2. Switches the activated disc to switch the turn using setTurn().
3. If getTurn() is blue, then setTurn() is used to switch turn to red and vice versa.
4. Calls the model and view class and updates them after the switch.

mouseClicked(MouseEvent e):

1. Called when the user presses a button, it will be responsible for handling the outcome of the button press.
2. Checks for mouse clicks during the game session (if the current state is the Main Menu, it returns null).
3. 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.
4. 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.

ConnectFourModel class contains the following public methods:

checkBoardConfiguration():

1. Checks the board configuration.
2. Uses boardConfiguration.length to check if theres a valid number of rows and columns.
3. If not, then prints error message, “invalid number of rows” or “invalid number of columns”.
4. Checks if the amount of blue discs and red discs are valid using the getRedDiscCount and getBlueDiscCount private methods with the newBoardConfiguration.
5. If not display "Too many blue discs" or "Too many red discs" depending on the error on custom game window.
6. Checks for floating discs by using a series of for loops and if statements.
7. If floating disc(s) is found display "Floating disc(s) detected" on custom game window.
8. Check for 4 discs in a row of the same colour using the private method.
9. If 4 discs are in a row display "Error four discs of the same colour together"

getRowSize():

1. Determines how many columns there are (the length of a row = number of columns).
2. Returns the length of the row, rowSize.

getWinState:

- This method checks whether or not the win state (4 red or 4 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.

saveState():

1. Saves the current state of the game into a text file, PATHTOSAVEDGAME.txt
2. Uses a forloop and a nested if statements to save the current state into the output file

loadState():

1. Loads the saved state of the game from the text file.
2. Input file is read and each disc is allotted the respective slot on the board.
3. While loop is used to read the input file and parse it to an array of strings.
4. Forloop is used to assign each index a slot on the board.

getGameProgress():

1. Returns the state of the current game.
2. If the game is in win state, GameProgress.redWon or GameProgress.blueWon, is returned.
3. If the number of blue discs and number of red discs equals the total number of slots, then its a tie game and GameProgress.tieGame is returned.
4. Else the game is still in progress and GameProgress.inProgress is returned.

*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.

getRandomTurn():

1. Sets the turn to either Red or Blue randomly and returns it.
2. Generates random number that is [1,3).
3. If random number is 1, then turn is set to Blue.
4. If random number is 2, then turn is set to Red.

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 to g.

getGameState()

- returns the current GameState

resetConfiguration()

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

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

nextAvailableSlot(int row, int col**):**

1. The slot where the users clicks on gives a point with (x,y) coordinates.
2. This method returns the coordinates of the first empty slot, from bottom to top, available in that column(where the user clicked).
3. That point is returned as Point(col, row).

*AI methods

Point doTurn():

1. Returns the position where the AI will place their piece.
2. Does this by calling on the nextAvaiableSlot() after a user turn.
3. While loop runs through all columns until it finds the next empty available slot.

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

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**)**

1. Creates the view object with the given arguments.
2. Width = the horizontal distance of the screen.
3. Height = the vertical distance of the screen
4. b = this is a 2d array that represents the state of each slot on the board. The state being either red, blue or empty
5. gs = the game state, for example mainMenu, CustomGame, Game

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

ConnectFourView()

- If no parameters are given, ConnectFourView() will call itself with the (default) parameters: initialScreenWidth, initialScreenHeight.

ConnectFourView(int, int**):**

- If only 2 parameters are given, ConnectFourView(int, int) will call itself with the first 2 given parameters, plus the (default) parameters: (int, int, 6, 7)

ConnectFourView(int, int, int, int**):**

1. Parameters:
  - **1.1.** <int> width of pane/window
  - **1.2.** <int> height of pane/window
  - **1.3.** <int> number of rows
  - **1.4.** <int> number of columns

2. The constructor creates the window. The parameters 1.1, 1.2, 1.3, and 1.4 are the width, height, number of rows, and number of columns respectively.
3. It will make the window visible and display the main menu.

4. It sets the default close option using the setDefaultCloseOperation(int operation) method.

5. Finally, it sets the size of the window size using the setSize(int width, int height) method.

switchScreen(ConnectFourModel.GameState gameState**):**

1. Responsible for the visibility of game panel and main menu depending on which state the game is in.
2. If game is in main menu then game panel is invisible (setVisible(false)) and vice versa.

paint(Graphics g):

1. This function draws the entire screen and updates all the buttons that need to be updated and resized
2. Redraws the board depending on the current GameState.
3. All buttons are repositioned and resized with the scale, when the screen is loaded.
4. g = the default Graphics that is used to draw on the screen.

setTurn(ConnectFourModel.Slot turn**)**

1. This function adjusts the images of the buttons so they change to their correct version to show the current button by highlighting it.
2. Updates the turn label by setting its turn (turn = the current turn such as blue or turn).

displayMessageAsPopup(String messageToDisplay**):**

- Displays and message on the screen of the text messageToDisplay that will be passed in as a parameter.

setMessageText(String error**)**

- Gets the string error message, sets it and displays it on screen.

addCalculateListener(ActionListener listenForButton, MouseInputListener listenForMouseEvents**)**

1. This method makes it the listener for the buttons the controller object.
2. Adds all the listener for the buttons. It gets the ActionListenr and MouseInputLister from the controller class.
3. listenForButton = the ActionListener that will respond to the button presses.
4. listenForMouseEvents = this is the mouse event listener that will respond to button presses even if they are not on the buttons.

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.

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 (Player vs Player, Player vs CPU, and Custom Game).

Board class contains the following public methods:

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

setBoard(ConnectFourModel.Slot[][] slots)

1. Gets the graphics components (getGraphics())and then draws the board.
2. Draws the board using that graphics component, drawTilesFromBoardConfiguration(getGraphics(), slots).

paintComponent(Graphics g)

- Draws the board if the controller exists using drawTilesFromBoardConfiguration(g, controller.getConfiguration())

setController(ConnectFourController c)

- sets the controller, controller = c

getGameCoordinate(Point slotPosition, int numberOfRows)

1. Converts the position with respect to the array to the position with respect to the game screen.
2. This function will take in the position of the board and return the actual position on the screen.

getBoardCoordinateOfPoint(Point mousePosition)

1. Converts the position to becoming with respect to the array instead of the game screen.
2. Convert the point mousePosition into a tile position where the x represents the column and y represents the row.

getWidthOfBoard():

- Returns Width of the board which is used to draw the board.

getHeightOfBoard():

- Returns height of the board which is used to draw the board.

isAnimating()

1. To tell the controller not to allow any other movements while the program is animating.
2. This is achieved by returning (animatingSlot != ConnectFourModel.Slot.Empty)

insertDisc(Point point, ConnectFourModel.Slot type)

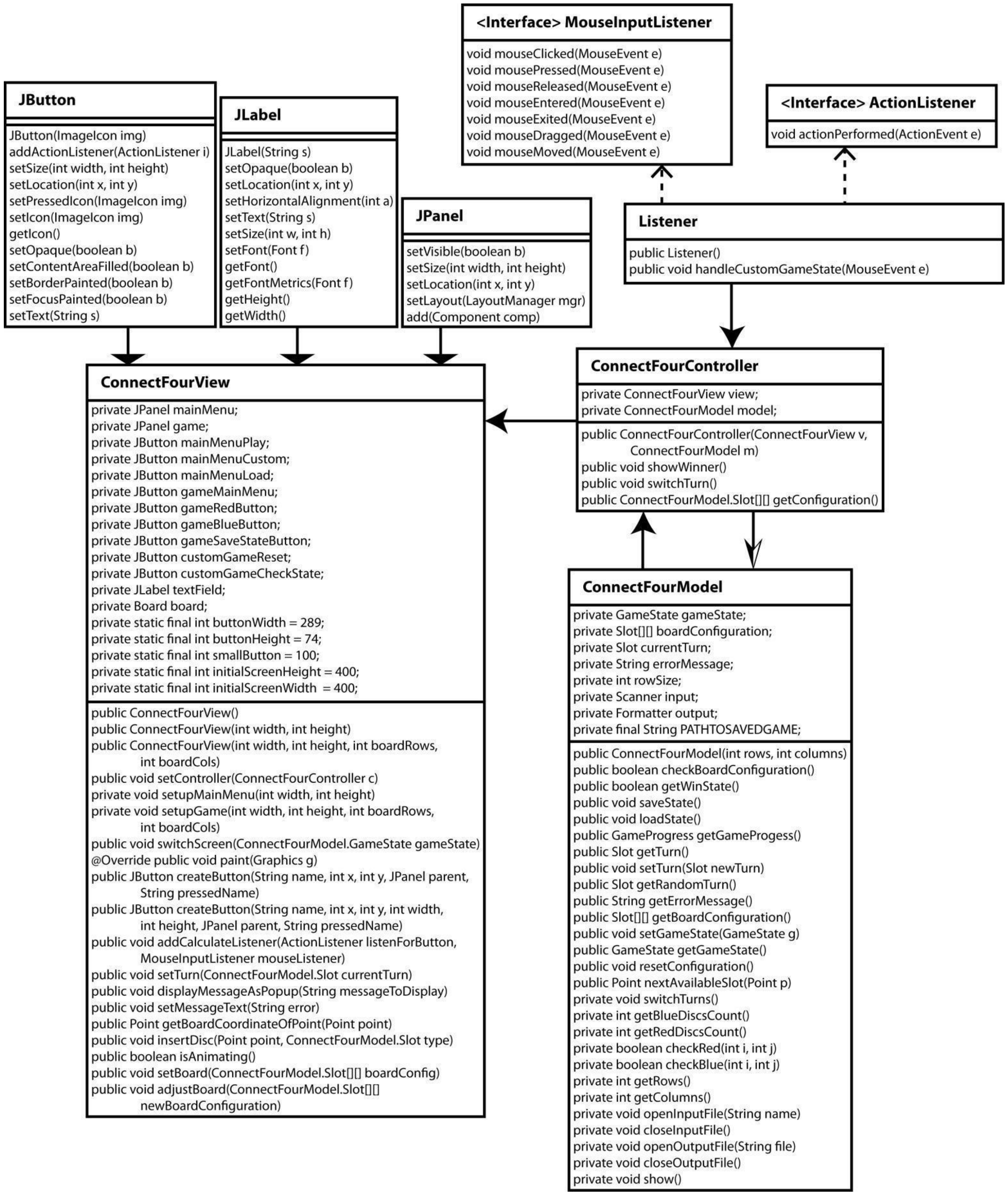
1. This function inserts the disc at the specified location.
2. The disc starts from the top row and then falls to the last row in the same column.

actionPerformed(ActionEvent e**):**

1. Called to animate the pieces fall.
2. It gets called every 100ms, it will update the animation variables, (animationPoint, stopAnimationPoint, animationSlot).

4.3 – Class Relation Diagram

TOP



<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)
<Interface> ActionListener
void actionPerformed(ActionEvent e)
JButton
JButton(ImageIcon img)
addActionListener(ActionListener i)
setSize(int width, int height)
setLocation(int x, int y)
setPressedIcon(ImageIcon img)
setIcon(ImageIcon img)
getIcon()
setOpaque(boolean b)
setContentAreaFilled(boolean b)
setBorderPainted(boolean b)
setFocusPainted(boolean b)
setText(String s)
JLabel
JLabel(String s)
setOpaque(boolean b)
setLocation(int x, int y)
setHorizontalAlignment(int a)
setText(String s)
setSize(int w, int h)
setFont(Font f)
getFont()
getFontMetrics(Font f)
getHeight()
getWidth()
JPanel
setVisible(boolean b)
setSize(int width, int height)
setLocation(int x, int y)
setLayout(LayoutManager mgr)
add(Component comp)
Listener
public Listener()
public void handleCustomGameState(MouseEvent e)
ConnectFourController
private ConnectFourView view;
private ConnectFourModel model;
public ConnectFourController(ConnectFourView v, ConnectFourModel m)
public void showWinner()
public void switchTurn()
public ConnectFourModel.Slot[][] getConfiguration()
ConnectFourView
private JPanel mainMenu;
private JPanel game;
private JButton mainMenuPlay;
private JButton mainMenuCustom;
private JButton mainMenuLoad;
private JButton gameMainMenu;
private JButton gameRedButton;
private JButton gameBlueButton;
private JButton gameSaveStateButton;
private JButton customGameReset;
private JButton customGameCheckState;
private JLabel textField;
private Board board;
private static final int buttonWidth = 289;
private static final int buttonHeight = 74;
private static final int smallButton = 100;
private static final int initialScreenHeight = 400;
private static final int initialScreenWidth = 400;
public ConnectFourView()
public ConnectFourView(int width, int height)
public ConnectFourView(int width, int height, int boardRows, int boardCols)
public void setController(ConnectFourController c)
private void setupMainMenu(int width, int height)
private void setupGame(int width, int height, int boardRows, int boardCols)
public void switchScreen(ConnectFourModel.GameState gameState)
@Override public void paint(Graphics g)
public JButton createButton(String name, int x, int y, JPanel parent, String pressedName)
public JButton createButton(String name, int x, int y, int width, int height, JPanel parent, String pressedName)
public void addCalculateListener(ActionListener listenForButton, MouseInputListener mouseListener)
public void setTurn(ConnectFourModel.Slot currentTurn)
public void displayMessageAsPopup(String messageToDisplay)
public void setMessageText(String error)
public Point getBoardCoordinateOfPoint(Point point)
public void insertDisc(Point point, ConnectFourModel.Slot type)
public boolean isAnimating()
public void setBoard(ConnectFourModel.Slot[][] boardConfig)
public void adjustBoard(ConnectFourModel.Slot[][] newBoardConfiguration)
ConnectFourModel
private GameState gameState;
private Slot[][] boardConfiguration;
private Slot currentTurn;
private String errorMessage;
private int rowSize;
private Scanner input;
private Formatter output;
private final String PATHTOSAVEDGAME;
public ConnectFourModel(int rows, int columns)
public boolean checkBoardConfiguration()
public boolean getWinState()
public void saveState()
public void loadState()
public GameProgress getGameProgess()
public Slot getTurn()
public void setTurn(Slot newTurn)
public Slot getRandomTurn()
public String getErrorMessage()
public Slot[][] getBoardConfiguration()
public void setGameState(GameState g)
public GameState getGameState()
public void resetConfiguration()
public Point nextAvailableSlot(Point p)
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 openInputFile(String name)
private void closeInputFile()
private void openOutputFile(String file)
private void closeOutputFile()
private void show()

4.4 TOP

ConnectFourView:
- When game is run, the view class creates a window with "New Game" and "Custom Game" buttons.
- Play Game has *Main Menu*, *Reset* and S*ave State* buttons displayed.
- 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 by highlighting the activated disc.
- Clicking on a spot on board draws the activated disc on the bottom most available row in that column.
- In Custom mode, if an already used spot on the board is clicked, the spot is erased if the activated disc is of the same color.
- In Custom mode, 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 Play Game mode, the user is redirected to the Main Menu.
- Custom Game redirects to Custom Game mode.
- Play Game redirects to Play 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:
- 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).

Board:
- Draws the board.
- Also used for the animation of gravity(falling pieces)

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

ConnectFourView Private Entities:

Instance Variables:

private JPanel mainMenu

- This is the panel that will be displayed while the game is in the main menu

private JPanel PvP

- This is the panel that will be displayed while the game is in either the actual Game or the Custom Game.

private JButton mainMenuPvP

- This button is located in the main menu. When pressed it will take the user to the Connect four Game.

private JButton mainMenuCustom

- This button is located in the main button. When pressed it will take the user to the Custom Game.

private JButton mainMenuLoad

- This button is located in the Main Menu. When pressed, it will load the saved state data and take the user to the Game.

private JButton mainMenuPvCPU

- button to go to the mode where you play against the computer.

private JButton gameMainMenu

- This button is located in the Game and in the Custom Game. When pressed, it will take the user to the Main Menu.

private JButton gameRedButton

- This button is located in the Custom Game. When pressed, it will set the current turn to Red's turn.
- If the current turn is already Red's turn, then there will be no change.

private JButton gameBlueButton

- This button is located in the Custom Game. When pressed, it will set the current turn to Blue's turn.
- If the current turn is already Blue's turn, then there will be no change.

private JButton gameSaveStateButton

- This button saves the state of a valid game.

private JButton customGameReset

- This button resets the board and makes it blank

private JButton customGameCheckState

- This button checks if the current state of the game is valid and then displays the error messages.

private JLabel textField

- This is the label that will display the error message in the custom game

private Board board

- This is the board that is displayed on the screen.

private static final int buttonWidth = 289

- Stores the width of all buttons

private static final int buttonHeight = 74

- stores the height of all the buttons

private static final int smallButton = 100

- stores the size of the small buttons

private static final int initialScreenHeight = 400

- these two constants store the dimensions of the screen

private static final int initialScreenWidth = 400

- if the client does not specify a value for the screen

Private Methods

private void setupMainMenu(int width, int height)

- sets up the main menu screen. It adds the corresponding components to the main menu panel

private void setupGame(int width, int height, int boardRows, int boardCols)

- Creates an empty panel and adds all the buttons and board to it.

<u>**ConnectFourController** Private Entities</u>:

Instance Variables:

private ConnectFourView view: The View class object used to update the view and notify the controller.

private ConnectFourModel model: The model class object used to update the model and notify the controller.

The ConnectFourController class contains no private methods.

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

ConnectFourModel Private Entities:

Instance Variables

private GameState gameState; Keeps track of the current state of the game (Game, Custom Game or Main Menu)
private Slot[][] boardConfiguration
- 2D array to represent the board configuration

private Slot currentTurn
- Represents the currently active colour state of the board.

private String errorMessage = ""
- The currently set error message to be displayed.

private int rowSize
- number of rows of the board.

private final String PATHTOSAVEDGAME = "data/savedGame.txt"
- Path to saved game.

Private Methods

private int getBlueDiscsCount()
- Count the number of blue discs inside the configuration

private int getRedDiscsCount()
- Count the number of red discs inside the configuration

private boolean checkRed(int i, int j)
- Check if certain slot has red disc

private boolean checkBlue(int i, int j)
- Check if certain slot has blue disc

private int getRows()
- Return number of rows

private int getColumns()
- Return number of columns

private void openInputFile(String name)
- Initialize the input file scanner

private void closeInputFile()
- Close the input file.

private void openOutputFile(String file)
- Opens the output file if it exists.

private void closeOutputFile()
- Closes output file

private void show()
- Visually see the board's configuration, used for debugging.

private int calculateScoreForTileAt(Point point)
- Calculates score for the AI so it can make the “smartest “ move.

private int findScoreAtDirection(int rowIncrementor, int colIncrementor, Point point)
- Used to calculate the score of a single position with a certain direction arguments.

private boolean isInBounds(Point point)

- Checks if the point that is passed in is inside the board or not.

private int getBoardWidth()

- returns the width of the board. Used to keep code abstract.

<u>**Board** Private Entities</u>:

Instance Variables

private int diameterOfDisk = 30

- represents the diameter of one disc

private int spaceBetweenDisks = 7

- represents the space between the disk

private int rows

- represents the number of rows in the board

private int columns

- represents the number of columns in the board

private ConnectFourController controller

- represents the controller

private Point animatingPoint

- used to animate the pieces.
- represents the position that the animation is currently on.

private ConnectFourModel.Slot animatingSlot

- used to animate the pieces.
- represents the color of the slot, if the current animating piece is red or blue.

private Point stopAnimationPoint

- the point at which the animation needs to reach when it stops animating
- represents the color of the slot, if the current animating piece is red or blue.

Private Functions

private void drawTilesFromBoardConfiguration(Graphics g, ConnectFourModel.Slot[][] slotConfiguration)

- calls the drawTileAtPosition in a nested loop to tell the drawTileAtPosition(); so all the tiles can be drawn if there is something to draw
- draws the board itself, which is just a black rectangle and then we will place the blank disks on top of it
- draws all the disks on the board

private void drawTileAtPosition(Graphics g, Point pos, ConnectFourModel.Slot type, int numberOfRows)

- draws one tile at the specified location
- draws the tile depending on the type and position

private Point getOriginOfBoard()

- gets the origin of the board meaning the bottom left coordinate of the board.
- Note the origin is the top left of the array/board and starts at slot 0.

4.5 TOP

Pros	Cons
-By implementing the MVC method, the robustness of the code is improved -The entire MVC is maintained fairly well, all the separate modules perform their specific functionality -View and Model not interacting made individual work easier as changes would not affect the other classes -The UI has been greatly improved from Assignment 1 and 2 -There is now a board class that contains methods previously found in the View to improve organization of the code and encapsulation -an efficient algorithm was used for the A.I as opposed to brute force -solutions to tasks presented in Assignment 1 and 2 are kept (still have a custom game state) so it retains previous features -The code is well documented and commented	-Our initial requirements could have been more specific, it would have made the task of coding far easier -We could have been more consistent between different classes and methods (some values have to be converted to be compatible between functions) -Inconsistent naming convention for documentation -Used swing but there are potentially better alternatives for this task -Could have added difficulty levels for the game versus the CPU

4.6 TOP

For the connect four game we could have used brute force to make the computer unbeatable but this method would be undesirable as it'd take too long to compute

The algorithm we made uses 3 steps.

1. The computer decides on the next move by accounting for all possible places where the computer can put a piece with a score. It will place the piece where it has the highest score.

2. To calculate the score the computer finds the score in each direction and adds them up.

3. To find the score in each direction the computer finds the number of pieces in that direction that have the same colour.

- A chain is a certain number of pieces with the same colour in any direction

- score = 2^(length of chain) for every chain. with the exception that if the length is lower then 2 then its 0 for length of 0 and 1 for length of 1

- the reasoning for this will be explained later

Examples

For all the examples we are going to use the letter

R for red piece

B for blue piece

E for empty

N for the piece can not be placed there when defining scores

only going to show the bottom elements

example:

B,R,E,E,E,E,E

refers to a board where there is 1 blue piece in the left most column and a red piece on the second left column

3:

Lets say in a specific direction we have the following

R, E, E, E, E, E. <-- pieces

N, 1, 0, 0, 0, 0, <-- scores given by step 3

E->1 = the score is 1 because there is a chain of 1 red and everything else is 0 or not available

R, R, E, E, E, E

N, N, 4, 0, 0, 0

E->4 = the score is 4 because 2^2 = 4

R, R, E, B, B, E

N, N, 8, N, N, 4

E->8 = the score is 8 because there are 2 chains of length 2 so it would be 2^2 + 2^2 = 4 + 4 = 8

The reason of why the score is calculated exponentially is because if we didnt then the AI would not block a chain of 3. For example

R, R, E, B, B, E

- the score for the first Empty would be 4 if score was calculated linearly

R, R, R, E, E, E

- the score for the first empty would be 3 if the score was calculated linearly.

by making the score be calculated by 2^(length of chain) it makes it so a chain of 3 has the highest priority no matter what is around it.

2:

This step adds the score of each directions and adds them up. So lets say we have a piece where in each direction their score is 2,4,8 the result will be 8+4+2 = 14

1:

In this step the AI will get the score for all the possible positions and chose the higest one. For example lets say there are 3 possible places to put the pieces and each of them have values 0, 3, 8. It will place the next piece where the value is 8.

4.7 TOP

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

Test Report TOP

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

Main Menu:

Game starts in Menu display	Check
Menu has three buttons: - Player vs Player - Player vs CPU - Custom Game - Load Save State	Check

Player vs Player:

Player vs Player switches to the game display with empty board and random player selected - Selected player is highlighted - Says “Game in Progress” - Has buttons for Main Menu, Reset and Save State	Check
Clicking on Main Menu switches back to Menu display	Check
Clicking on the board: - Animates the selected color into the bottom most spot of the clicked column - Switches to the other color and highlights it	Check
Clicking Reset clears the board	Check
Clicking Save State checks the board and saves the state if it is valid	Check

- Shows an appropriate pop up message	
Winning: - If four blue/red in a row are detected, it says “blue/red won” - Board gets disabled - Save state doesn’t work - In case no one wins, says “tie game” (and does the same stuff listed above)	Check

Player vs CPU

*This retains much of the same functionality of Player vs Player so only additional CPU specific tests are listed below

Player and CPU turns alternate one after the other.	Check
CPU takes its turn in a timely matter keeping the game flowing smoothly.	Check
CPU attempts to place 4 pieces in a row and does so if the player does not block it.	Check
CPU blocks the player's winning move.	Check
If the board is full the game ends and the CPU does not continue to place pieces on the board.	Check.

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

Custom Game:

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
Clicking Save State checks the board and saves the state if it is valid - Shows an appropriate pop up message	Check

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

Load Save State:

Loads most recently saved configuration in the New Game display	Check
Takes into account the person whose turn it was while saving the game	Check