Connect Four Assignment 4

Group Name: Group 20 (D-Bugs) **Group Members:** Aamir Badruddin
Hejie Yan
Xiaoying Wang
Mario Calce
Yuchen Liu
Mingfei (David) Jiang

The game will be composed of three classes: `Board`, `BoardController` and `BoardData`. With the requirements for this assignment, three classes were decided upon to meet the requirements of the game, BoardData was decided to contain the model of the game, including the logic and conditionals that runs the board "internally". Board was chosen to display the graphical user interface of the board, it acquires information from the model and ouputs it to the user, the board also implements actionlisteners to await input from the user to display counters onto the board where the user has indicated with mouse clicks. BoardController class is used so that the user can interact with the Board. *The game is able to run on computers with Java installed.*

***Board.java**

The Board class will contain the main function which initializes object variables, and runs a method to create the board, listens for player actions, and prints out tokens to the board based on these actions.

***Access Programs**

`*main()`

- *Initialize BoardGUI variable
- *Modify the BoardGUI's properties
- *Determines whether game is against a player or a computer*
-

`*BoardGUI()`

- Initialize all variables *for displaying the board*
- Create menu items:
    - Reset
    - Save
    - load
    - Instructions
    - Quit
- Display whose turn it is
- Represent the board with 2D array

Contained within class BoardGUI:

`*BoardGUI()` *`Contructor`*

- Should be set up inside public static class BoardGUI
- Set up menu bar
- Put together buttons and board
- Set up actionListeners for buttons
- *Creates the grid to display the board in the View*

`disableAll()`

- Should disable all column buttons on the board

`buttonEnable()`

- *enables all buttons in the GUI*

`refresh()`

- *refreshes the click counter to reflect how many*
-

`winner(String winningColour)`

- Disables all buttons, then displays who the winner is

`actionPerformed()`

- Add chip to corresponding column and switch turn on display
- Display winner, if one is found

`SaveListener()`

- *Save state of variables of actionPerformed()*

`LoadListener()`

- *Loads state of all variables if a game was saved in the session*

`InstructListener()`

- Display instructions

`QuitListener()`

- *Closes the application*

`ResetListener()`

- Resets the board

***Implementation**

***Access Programs**

`*main()`

- *Initialize BoardGUI object
- *Creates a BoardController object, "controller"*
- *Modify the BoardGUI's properties:
    - Set the title to Connect Four
    - Set it as visible
    - Set the size at a default 500x500
    - Set the close operation to JFrame.EXIT_ON_CLOSE

`*BoardGUI()`

- public static class should extend JFrame and implements ActionListener
- Initialize 7 private ints click counters for tracking the number of clicks per column, and set them all equal to 0
- Initialize 7 private JButtons for each column
    - Set text to "Add" for each
- Create menu items using private JMenuItem for:
    - Reset
    - Save
    - load
    - Instructions
    - Quit
- Display to the user which player's turn it is using JLabel and and a getTurn method from the controller
- Create 6 by 7 array of JLabel cells, to represent the board
- Initialize private Color variable

`*BoardGUI() Constructor`

- Set up menu using JMenu and JMenuBar
    - Actions should hold Reset, Save, Load and Quit
    - Help should hold Instructions
- Create JPanel, 1 by 1, to hold the JButtons for columns
- Create another JPanel, 6 by 7, for the board
- Use 2 for loops to set up properties of every entry in the 6 by 7 array cells:
    - Set JLabel to ""
    - setBorder(BorderFactory.createEtchedBorded())
    - setOpaque(true)
    - Add each to cells to the board JPanel
- Set Color variable to cells[0][0].getBackground()

- Initialize a container to getContentPane(), and add:
    - The board panel
    - The buttons panel
    - The JLabel that displays whose turn it is
- Add actionListeners to every column JButtons and JMenuItems
- Create container to add all actionlisteners

`disableAll()`
- *For each column button, setEnabled(False)*

`buttonEnable()`
- *For each column button, setEnabled(True)*

`refresh()`
- *double-loop to cycle through each label, for each label that is has a token in it, add one to the column button click counters*

`winner(String winningColour)`
- Calls disableAll method
- Use showMessageDialog to display which colour the winner is

`actionPerformed()`
- Set up cases for each of the 7 Column button presses
- For each column case:
    - add a chip to controller by calling .add(Column)
    - Set background of cell[row][column] to red/blue depending on who placed it
        - Row will be found with one of the 7 private int initialized at the beginning for tracking the row
        - Column relates to the button pressed
    - Increment row variable by 1
    - Set case that if row == 6, disable Column JButton
- Update the text to display whose turn it is
- Set case that if winner is found, display which player is the winner

`SaveListener()`
- *Prompts user input for Name of Save File*
- *Calls to save function in controller Controller object with parameters for title, and player turn*

`LoadListener()`
- *JFileChooser object loadGame*

- *calls to controller.load(filename), if JFileChooser approves the file*
- *Use double for-loop to cycle through grid to update the click counter of the column buttons*
- *Display text for current player turn*

`QuitListener()`

- *Waits for action to be performed → System.exit(0);*

`InstructListener()`

- Should be set up inside public static class BoardGUI
- Use showMessageDialog to show instructions

`ResetListener()`

- Should be set up inside public static class BoardGUI
- Resets the board
- Implement ActionListener
- Use 2 for loops to iterate throught the board and for each cell:
    - Set text to “”
    - Set background to default color (saved from BoardGUI)
    - Set border to BorderFactory.createEtechedBorder()
    - Set opaque
    - Call reset on controller
- Reset all columns counters to 0
- Enable all column JButtons
- Reset turn and display

***Class: BoardData**

Package connectFour

The BoardData class contains methods and logic to add discs on the board, reset the game to initial state, and check the winner of the game. It construct a 2-D array `data` that its data type is string. The BoardData class contains following methods:

Access Programs

Import:

*java.util.**
*java.io.**
javax.swing.JOptionPane

`BoardData()`

- Create a matrix representation of the game board
- It is a constructor and the matrix is **7 by 6**

`BoardData Copy()`

- Copy the data from the board and saves it to be loaded at a future time
- Returns the new BoardData `d`

`AddRed(int i)`

- Add red discs on the board

`AddBlue(int i)`

- Add blue discs on the board

`GetColour(int col, int row)`

- return the contents (“Red” or “Blue”) of the specified cell

`Reset()`

- Clear the discs on the board and reset it to original state

`isWin()`

- Check which user is win
- It checks the board horizontally, vertically, and diagonally

`*save()`

- Save the current game

`*load()`

- Load the saved game

`stateToString()`

- *Change the number at specific cell on board matrix from type `Integer` to type `String`*

***Implementation**

Import:

*java.util.**
*java.io.**
javax.swing.JOptionPane

***Variables**

`data: String[][]`

- Stored a 7 by 6 matrix for the game board

***Access Programs**

`BoardData(): String[][]`

- Use a nested for loop to create a 2-D array
    - The data type of cells is string, and initially contains nothing

`BoardData Copy(): String[][]`

- It create a new BoardData `d` which is the copy of BoardData
- Use a nested for loop

`AddRed(int i)`

- Use a while loop to get the location of where the red discs are added
- A variable counter increments in the while loop and used to indicate the location of red discs

`AddBlue(int i)`

- Use a while loop to get the location of where the blue discs are added
- A variable counter increments in the while loop and used to indicate the blue discs' location

`GetColour(int col, int row): String`

- Returns the content (colour of of the disc) at specific cell

`Reset(): String[][]`

- Uses a nested for loop to set the contents of each cell to " "

`isRWin(): String`

- It checks which user is win
- Returns String `winner`
- Vertical Check:
    - for col = 0 to 6, do:
        for row = 0 to 2, do:
            if data[row][col] == data[row+1][col] and data[row][col] == data.[row+2][col] and data[row][col] == data[row+3][col] and data[row][col] == “red”, then: winner = “red”
            if data[row][col] == data[row+1][col] and data[row][col] == data.[row+2][col] and data[row][col] == data[row+3][col] and data[row][col] == “blue”, then: winner = “blue”
- Horizontal Check:
    - for row = 0 to 5, do:
        for col = 0 to 3, do:
            if data[row][col] == data[row][col+1] and data[row][col] == data.[row][col+2] and data[row][col] == data[row][col+3] and data[row][col] == “red”, then: winner = “red”
            if data[row][col] == data[row][col+1] and data[row][col] == data.[row][col+2] and data[row][col] == data[row][col+3] and data[row][col] == “blue”, then: winner = “blue”

- Diagonal Check:
    - for row = 0 to 2, do:
        for col = 0 to 3, do:
            if data[row][col] == data[row+1][col+1] and data[row][col] == data.[row+2][col+2] and data[row][col] == data[row+3][col+3] and data[row][col] == “red”, then: winner = “red”
            if data[row][col] == data[row+1][col+1] and data[row][col] == data.[row+2][col+2] and data[row][col] == data[row+3][col+3] and data[row][col] == “blue”, then: winner = “blue”
    - for row = 3 to 5, do:
        for col = 0 to 3, do:
            if data[row][col] == data[row-1][col+1] and data[row][col] == data.[row-2][col+2] and data[row][col] == data[row-3][col+3] and data[row][col] == “red”, then: winner = “red”
            if data[row][col] == data[row-1][col+1] and data[row][col] == data.[row-2][col+2] and data[row][col] == data[row-3][col+3] and data[row][col] == “blue”, then: winner = “blue”

stateToString()

- Use a nested for loop to reach the specific location on the board matrix
- Use the toString method to change content from type integer to type string

Class: BoardController.java

***Access Programs**

`BoardController class`

- Initialize variables

`*whoOnFirst()`

- Determine which player goes first

`*getTurn()`

- Returns which player's turn it is

`*changeTurn()`

- Switches turn

`*getColour(int row, int column)`

- Returns the background colour of the cell at the specified row and column

`*reset()`

- Reset the board and re-rolls random number to determine who goes first

`*add(int column)`

- Add blue/red token to specified column

`*winner()`

- Returns who won

`*save(String saveFile,String turn)`

- Saves token positions into a text file

`*load(String gameFile)`

- Loads and displays token positions from text file

***Implementation**

***Access Programs**

`BoardController class`

- Initialize String variable turn, for determining which player's turn it is
    - Run whoOnFirst() to determine whose turn it is, and store in variable
- Initialize BoardData board

`*whoOnFirst()`

- Determine which player goes first
- Use Math.random() to determine whether red or blue goes first
- Returns string “blue” or “red”

`*getTurn()`
- Returns which player’s turn it is

`*changeTurn()`
- Switches turn
- Use if case from red->blue or blue->red
    - change turn value to red/blue

`*getColour(int row, int column)`
- Returns the background colour of the cell at the specified row and column
- Using row and column, return colour using board.GetColor(row, column)

`*reset()`
- Reset the board and re-rolls random number to determine who goes first
- Call board.reset()
- Call whoOnFirst() for first player

`*add(int column)`
- Add blue/red token to specified column
- Use if/else to determine which colour is placing a token
- Add blue/red token to column using board.AddBlue(column) or board.AddRed(column)
- Change turn using this.changeTurn()

`*winner()`
- Returns who won
- Ask the model if there is a winner using board.isWin() ==”blue” or board.isWin()==”red”
- Use if/else statement to determine who the winner is
- Return winner string as “blue” or “red”

`*save(String saveFile,String turn)`
- Saves token positions into a text file
- Call `board.save` with parameters `String saveFile` and `String turn`

`*load(String gameFile)`
- Loads and displays token positions from text file
- Call `board.load` with parameter `String gameFile`
- Modify controller context turn by calling board.getTurn
    - `this.turn = board.getTurn();`

Class: GameAI.java

Access Programs

`getMove()`

- *returns field variable move*

`move()`

- *generates random movement for AI*

`setMove()`

- *set move to be used by external method*

Implementation

- *private int move;*

`getMove()`

- *return move;*

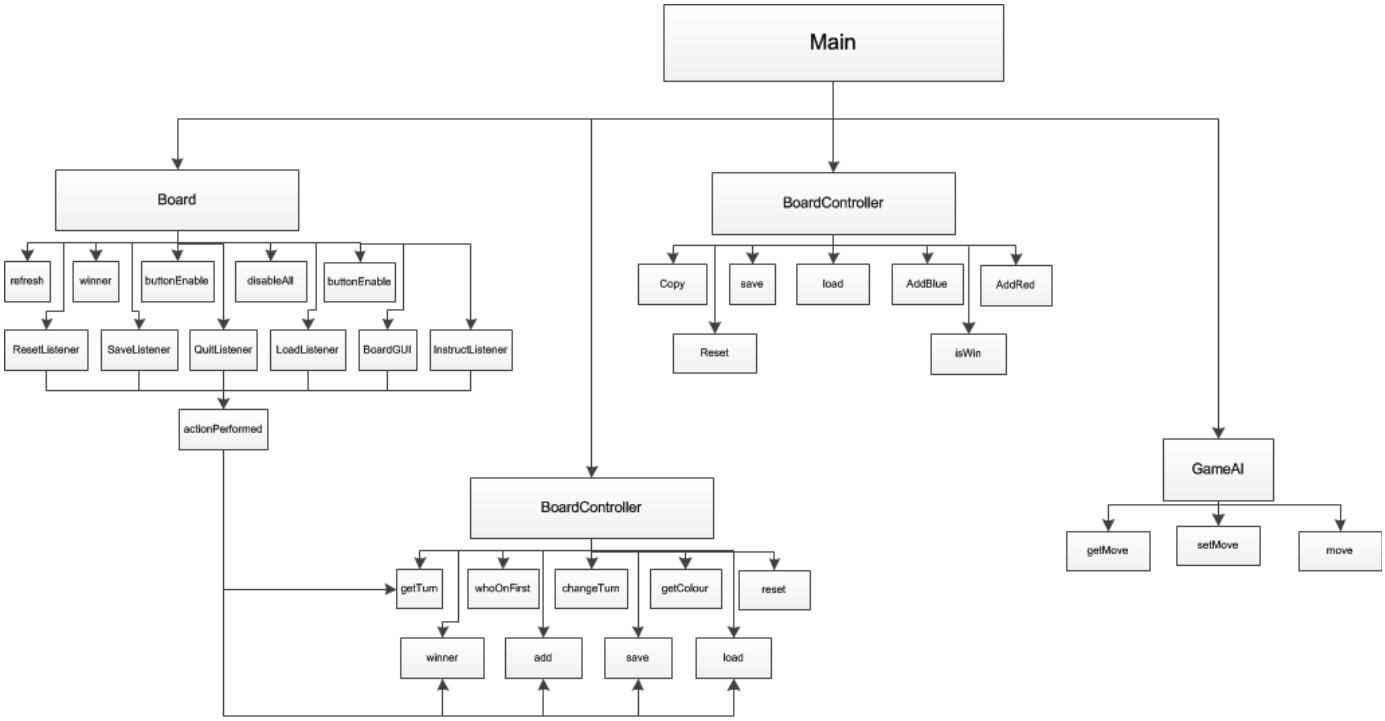
`setMove()`

- *move();*

`move()`

- *Random object rand*
- *sets move field to rand.nextInt(6)+1*

*View of Uses-Relationships



***Internal Review/Evaluation of Design:**

The program meets the requirements in assignment 3. We added one more class called GameAI on top of three classes, which is simple AI allows player to play again computer. The visual flaws from the second version (Assignment 3) has been improved so that it is much more clean and appealing to users. A window pops up at the beginning of the game that asks the player to choose the mode of the game. The load, save, quit and reset functions have either been improved or added. However, the AI should be improved since it only follows very simple rules. Player has much more higher winning rate than AI. More scenarios should be considered when implementing the AI.

***Traceability:**

Methods	Assignment 3 Requirement										
	1	2	3	4	5	6	7	8	9	10	11
main()	✓										
BoardGUI()	✓										
InstructListener()									✓		
actionPerformed()				✓	✓						
disableAll()										✓	
winner()										✓	
ResetListener()			✓								
BoardData()	✓										
Copy()											
AddRed()				✓							
AddBlue()				✓							
GetColour()				✓							
Reset()			✓								
isWin()						✓			✓		
save()							✓				
load()								✓			
whoOnFirst()		✓									
getTurn()				✓						✓	
changeTurn()				✓	✓					✓	
getColour()				✓							
reset()			✓								
add()				✓	✓						

getMove()											✓
setMove											✓
move()											✓

* Requirements:

1. Set up a 7 by 6 game board with menu option
2. Randomly determine the order of play
3. Choose to start a new game
4. Place different colour discs on board
5. Make moves and report error if the move is illegal
6. Determine the winner of the game
7. Save games
8. Load the saved games
9. Show messages
10. Show game status
11. AI

***Testing**

Input	**Expected Output**	**Output**	**Result**
Game mode selection: **Player vs Player**	2 player game mode is started	2 player game mode is started	**PASS**
Game mode selection: **Player vs Computer**	Player vs AI game mode is started	Player vs AI game mode is started	**PASS**
Add button to **empty column**	● Coloured token is dropped into next available position on the column ● Player turn switches and is displayed	● Token is dropped depending on player's turn ● Player turn changes properly	**PASS**
Add button to **full column**	Turn not accepted	Player turn is not changed	**PASS**
Add token to **diagonal winning condition**	Checks winning condition for diagonal	Dialog box pops up with winning player	**PASS**

	Displays winning player		
Add token to **horizontal winning condition**	Checks winning condition for horizontal	Horizontal winning condition is checked and winning player is displayed on dialog pop up	**PASS**
Add token to **vertical winning condition**	Checks winning condition for vertical	Vertical winning condition is checked and winning player is displayed on dialog pop up	**PASS**
Board is full of tokens	Shows player that game has ended	Does not do anything	**FAIL**
Actions>**Reset**	Resets board of all tokens Resets player turn randomly New game is playable	Rests board of all tokens Resets player turn randomly New game is unplayable	**FAIL**
Actions>**Save**	Saves tokens' positions in a separate text file	Saves tokens' positions in a text file at the application directory	**PASS**
Actions>**Load**	Loads token positions from text file	Does not load previous state	**FAIL**
Actions>**Quit**	Quits the app	Quits	**PASS**
Help>**Instructions**	Shows instructions in separate dialog box	Shows instructions in separate dialog box	**PASS**