Tricky five-in-a-row

Nazifa Nazrul Rodoshi

W8CZNE

Contents

Task

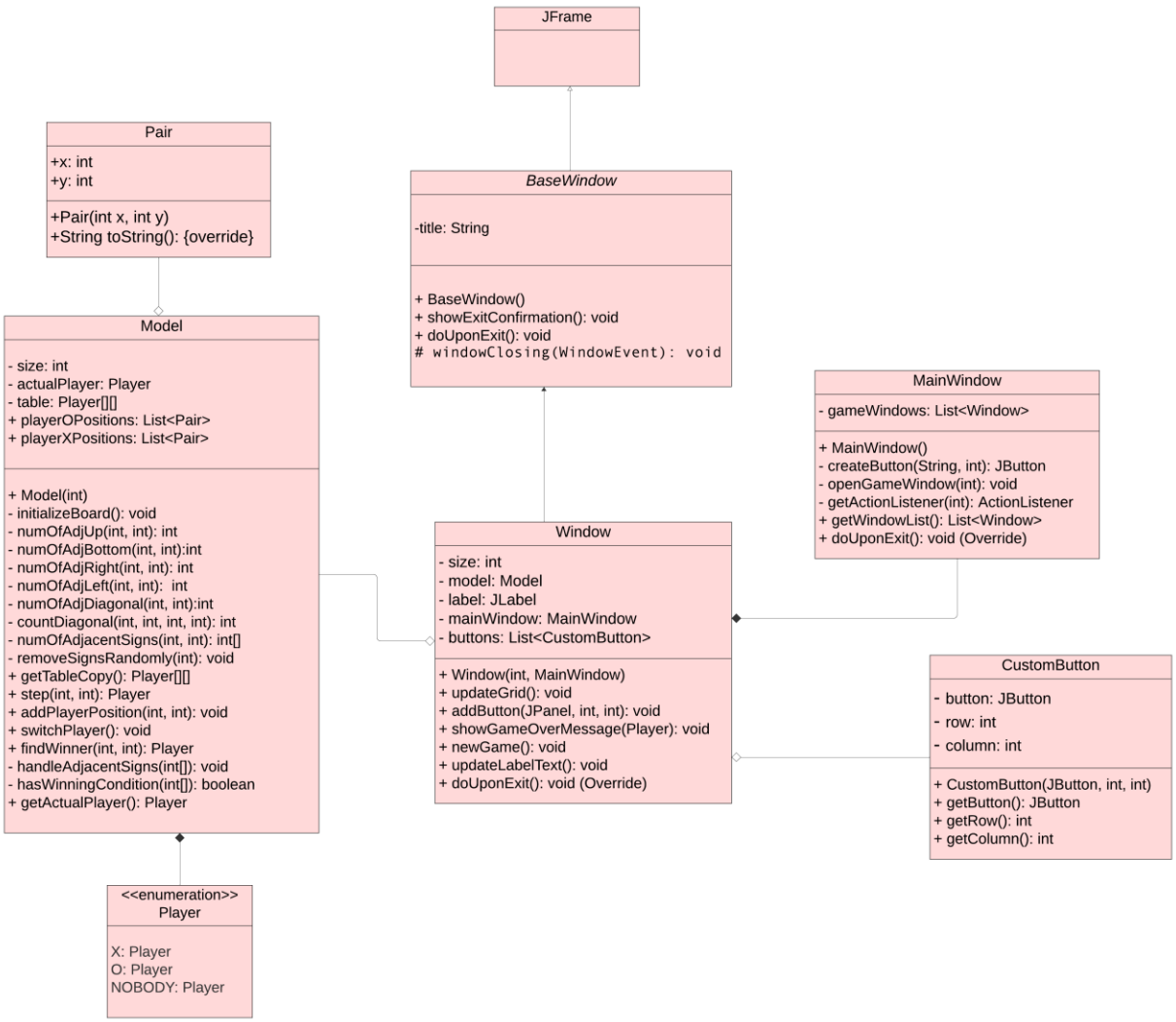
Create a game, which is a variant of the well-known five-in-a-row game. The two players can play on a board consists of n x n fields. Players put their signs alternately (X and O) on the board. A sign can be put only onto a free field. The game ends, when the board is full, or a player won by having five adjacent signs in a row, column or diagonal. The program should show during the game who turns. The trick in this variant is that if a player makes 3 adjacent signs (in a row, column or diagonal), then one of his signs is removed randomly (not necessary from this 3 signs). Similar happens, when the player makes 4 adjacent signs, but in this case two of his signs are removed. Implement this game, and let the board size be selectable (6x6, 10x10, 14x14). The game should recognize if it is ended, and it has to show in a message box which player won (if the game is not ended with draw), and automatically begin a new game.

Task Description

GUI classes that set the user interface part of the game are: Window.java, MainWindow.java and BaseWindow.java.

The logic of the game is set in class Model.java. the main class is in TrickyFiveInRow.java.
Other classes are : Pair.java, Player.java and CustomButton.java

UML Diagram



Description of the methods

Class Pair

- Pair(int x, int y): Constructor that initializes a Pair object with coordinates x and y.
- toString(): Returns a string representation of the Pair object in the format "(x,y)".

Class BaseWindow

- Constructor BaseWindow(): Initializes the window with a title, size, custom close operation, and an icon.
- Method showExitConfirmation(): Displays a confirmation dialog when closing the window.
- Method doUponExit(): Closes and disposes of the window.

Class CustomButton

- Constructor CustomButton(JButton button, int row, int column): Initializes a CustomButton with a specified JButton, row number, and column number.
- Method getButton(): Returns the JButton associated with this CustomButton.
- Method getRow(): Returns the row number where this button is located.
- Method getColumn(): Returns the column number where this button is located.

Class Window

- Constructor Window(int size, MainWindow mainWindow): Initializes the game window with a grid of buttons, label, and new game button.
- Method updateGrid(): Updates the grid's buttons based on the current state of the game model.
- Method addButton(JPanel panel, int row, int column): Adds a button to the grid and sets its action listener to update the game model and UI.
- Method showGameOverMessage(Player winner): Displays a message dialog announcing the game winner and starts a new game.
- Method newGame(): Creates a new game window and disposes of the current one.
- Method updateLabelText(): Updates the text of the label to show the current player.
- Override doUponExit(): Removes the window from the main window's list and performs the parent class's exit procedure.

Class Model

- Constructor Model(int size): Initializes the game model with a specified size, sets the starting player, and initializes the game board.
- Method initializeBoard(): Fills the game board with Player.NOBODY to represent an empty state.
- Methods numOfAdjUp, numOfAdjBottom, numOfAdjRight, numOfAdjLeft: Count the number of adjacent cells in the specified direction that are occupied by the current player.
- Method numOfAdjDiagonal(int row, int column): Calculates the maximum number of adjacent diagonal cells occupied by the current player.
- Method countDiagonal(int row, int column, int rowIncrement, int colIncrement): Counts the number of adjacent diagonal cells in a specific diagonal direction.
- Method numOfAdjacentSigns(int row, int column): Returns an array with counts of adjacent cells in up, right, left, bottom, and diagonal directions.
- Method removeSignsRandomly(int numberOfSigns): Removes a specified number of signs from the current player's positions randomly.
- Method getTableCopy(): Returns a copy of the current game board.
- Method step(int row, int column): Processes a move at the specified row and column, updates the game state, and returns the player occupying that cell.

- Method addPlayerPosition(int row, int column): Adds the current move to the list of positions for the player who made the move.
- Method switchPlayer(): Switches the turn to the other player.
- Method findWinner(int row, int column): Determines if there is a winner after the recent move.
- Method handleAdjacentSigns(int[] adjacentSigns): Handles game logic based on the number of adjacent signs.
- Method getActualPlayer(): Returns the player who is currently making their move.

Class FiveInRow

- Method main(String[] args): Creates an instance of MainWindow and makes it visible, effectively starting the Five-In-A-Row game application.

Class MainWindow

- Constructor MainWindow(): Sets up the main window with buttons for different game sizes (6x6, 10x10, 14x14) and adds them to a panel.
- Method createButton(String text, int size): Creates and returns a button with specified text and size, and attaches an action listener to it.
- Method openGameWindow(int size): Opens a new game window of the specified size.
- Method getActionListener(final int size): Returns an ActionListener that creates and opens a new game window of the given size when triggered.
- Method getWindowList(): Returns the list of currently open game windows.
- Override doUponExit(): Terminates the application when the main window is closed.

Event-Handler Connections

Button Creation (MainWindow):

- Method: createButton
- Function: Creates buttons for different game sizes and attaches action listeners to them.

Opening Game Window (MainWindow):

- Event: Button press in MainWindow.
- Action: Invokes openGameWindow, opening a new game window with the specified grid size.

Game Grid Initialization (Window):

- Constructor logic in Window.
- Function: Initializes a grid of CustomButtons, attaching action listeners to each.

Button Press Handling (Window):

- Event: Pressing a CustomButton in a game window.
- Action: Updates game model and state, specific details depend on the implementation in the button's action listener.

Game State Check (Window):

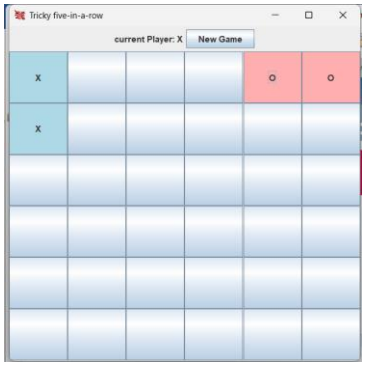
- Part of button press handling.
- Function: Checks game state after each button press (e.g., win conditions, next player's turn).

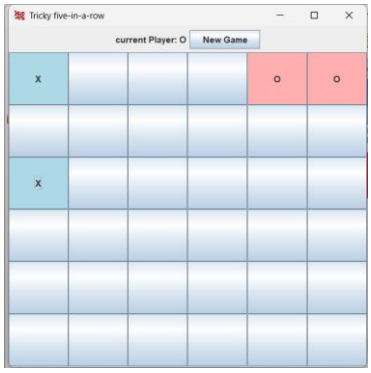
New Game Initiation (Window):

- Event: Pressing the 'New Game' button.
- Action: Triggers the newGame method, resetting the game for a new start.

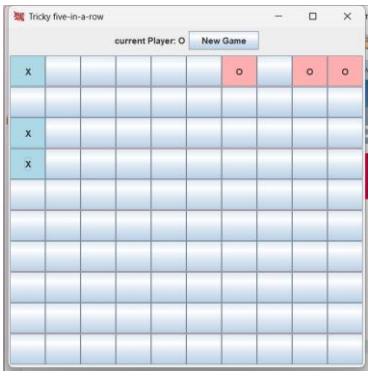
Test

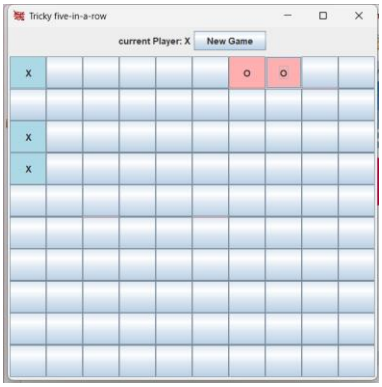
1. **Remove 1 Test**
    - Objective: Test if it removes 1 sign of same type as current player when 3 same signs are together.
    - Expected Result**:** works successfully.



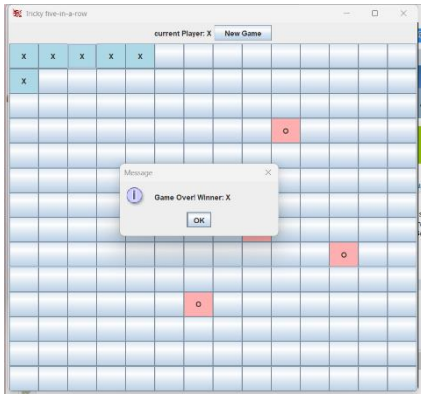


2. **Remove 2 Test**
    - Objective: Test if it removes 2 signs of same type as current player when 4 same signs are together.
    - Expected Result: works successfully.

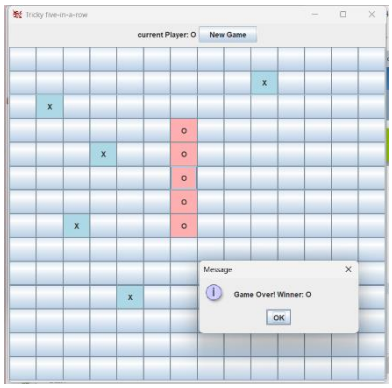




3. **Winning Horizontal Test**
    - Objective: Test if player wins when 5 same signs are together horizontally.
    - Expected Result: wins the game.



4. **Winning Vertical Test**
    - Objective: Test if player wins when 5 same signs are together vertically.
    - Expected Result: wins the game.



5. **Closing Test**
    - Objective: Tests if quitting the game also disposes the game terminal.
    - Expected Result**:** successfully terminates upon closing the game.