Programming Technology Documentation

Otar Jinchveladze

November 2024

Contents

1 Introduction

This document serves as the official documentation for the programming technology assignment. It outlines the goals and structure of developing as well as implementing the project. The primary objective of this assignment is to enhance our understanding of object-oriented programming principles. Additionally improve problem-solving skills through the design of a simulation-based game.

This document is created by Otar Jinchveladze, a second-year student pursuing computer science at Eötvös Loránd University (ELTE). This document will be reviewed by my teacher, Guettala Walid. He is a PhD student at ELTE with a background in deep learning and robotics, currently researching Graph Neural Networks.

Throughout this documentation, I will cover various aspects of the project. Including detailed descriptions of the exercise and the implementation of methods. Moreover, testing procedures and results will be provided to demonstrate the functionality and reliability of the program. All this presents an opportunity to gain hands-on experience in designing modular, maintainable code and applying core programming concepts in a practical scenario.

2 Description of the Exercise

2.1 Exercise 4

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.

3 Class Diagram

The class diagram for the Tricky five-in-a-row game provides a comprehensive overview of the system architecture and the relationships between different components of the game. It is designed to illustrate the key classes, their attributes, methods, and the interactions among them.

Classes: BaseWindow, Window, MainWindow (Subclasses of BaseWindow). Model, Player (enum).

3.1 Main Class Diagram

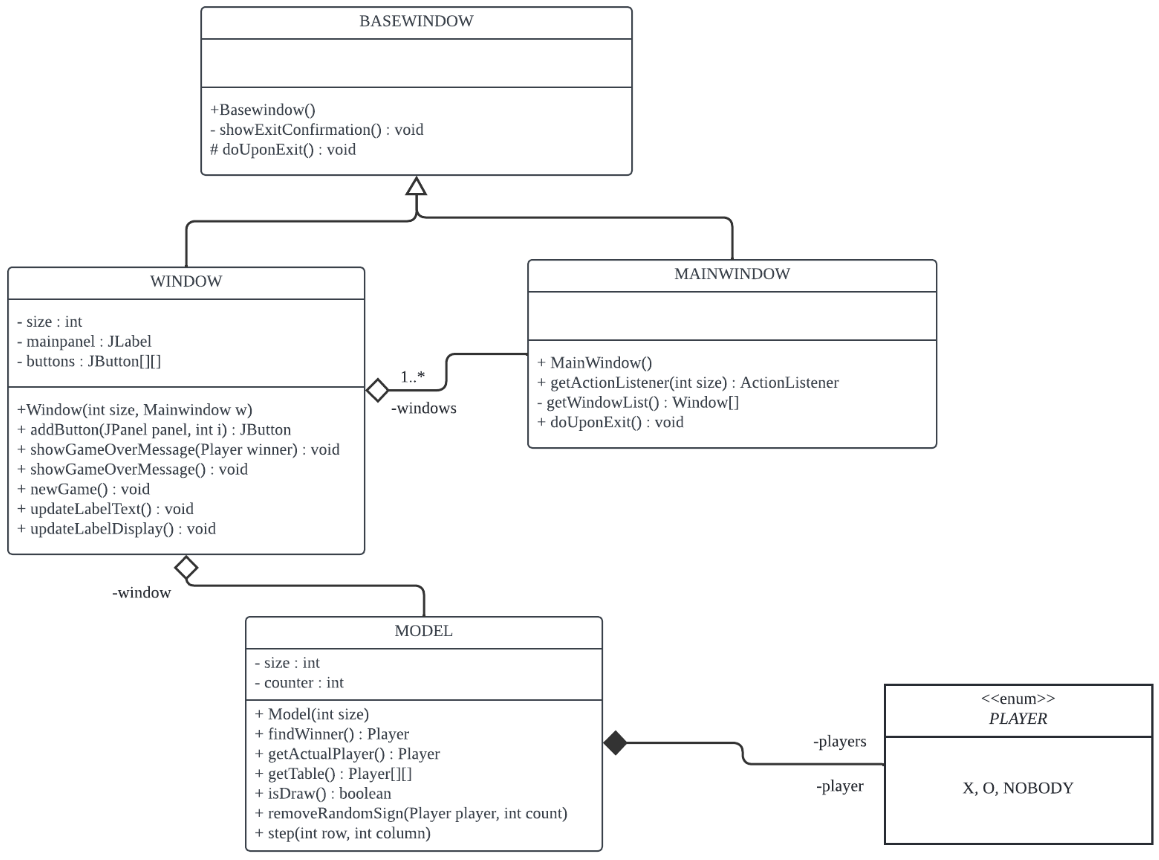
BASEWINDOW
+Basewindow()
- showExitConfirmation() : void
# doUponExit() : void
WINDOW
- size : int
- mainpanel : JLabel
- buttons : JButton[][]
+Window(int size, Mainwindow w)
+ addButton(JPanel panel, int i) : JButton
+ showGameOverMessage(Player winner) : void
+ showGameOverMessage() : void
+ newGame() : void
+ updateLabelText() : void
+ updateLabelDisplay() : void
MAINWINDOW
+ MainWindow()
+ getActionListener(int size) : ActionListener
- getWindowList() : Window[]
+ doUponExit() : void
1..*
-windows
-window
MODEL
- size : int
- counter : int
+ Model(int size)
+ findWinner() : Player
+ getActualPlayer() : Player
+ getTable() : Player[][]
+ isDraw() : boolean
+ removeRandomSign(Player player, int count)
+ step(int row, int column)
-players
-player
<<enum>>
PLAYER
X, O, NOBODY

Figure 1: Class Diagram for the Tricky five-in-a-row game

4 Description of Methods

This documentation will not include getter and setter methods, as they are straightforward and commonly understood for managing access to private and protected fields. Getters and setters serve a clear purpose: getters allow external classes to access the values of private fields, while setters enable them to modify those values. Since these methods are standard practices in object-oriented programming and their functionality is generally familiar to developers, we will focus on documenting the more complex methods and functionality of the classes. This approach helps streamline the documentation by omitting repetitive explanations and allowing us to concentrate on the unique features and behaviors of the system that may require further clarification or context.

4.1 BaseWindow Class

4.1.1 Method: showExitConfirmation()

The showExitConfirmation method displays a confirmation dialog asking the user if they want to quit. It provides "Yes" and "No" options. If the user selects "Yes," the method calls doUponExit() to handle any exit procedures. This method helps ensure the user confirms before exiting.

4.1.2 Method: duUponExit()

The doUponExit method is a protected method that closes the current window or application. It calls this.dispose(), which releases the resources associated with the window and removes it from the screen. This method is typically used as part of an exit process to ensure a clean shutdown.

4.2 Window Class

4.2.1 Method: addButton(JPanel panel, int i)

Creates a new button for the board at position (i, j), adds it to the specified panel, and assigns an action listener. When clicked, the button triggers a game step and updates the board display.

4.2.2 Method: showGameOverMessage(Player winner)

Displays a message dialog announcing the winner and starts a new game.

4.2.3 Method: showGameOverMessage2()

Displays a "Draw" message when the game ends in a tie and initiates a new game.

4.2.4 Method: newGame()

Starts a new game by creating a new Window instance and disposing of the current one. It also removes this window from the main window's list.

4.2.5 Method: updateLabelText()

Updates the label to display the current player's name.

4.2.6 Method: updateLabelDesign()

Updates each button on the board to reflect the current state, displaying the player's symbol or clearing the button if unoccupied.

4.3 MainWindow Class

4.3.1 Method: getActionListener(int size

Returns an ActionListener for creating a new game window with the specified board size. When triggered, it opens a new Window instance with the given size.

4.3.2 Method: getWindowList()

Returns the list of active game windows, allowing other parts of the program to access and manage open windows.

4.3.3 Method: doUponExit()

Override: Overrides the exit method to terminate the application when the main window is closed.

4.4 Model Class

4.4.1 Method: findWinner()

Checks the board for a winning line of five consecutive marks in any direction. If there are fewer consecutive marks, it may remove random marks from the board to add complexity.

4.4.2 Method: getActualPlayer()

Returns the current player whose turn it is.

4.4.3 Method: getTable()

Returns the current state of the game board as a 2D array of Player values.

4.4.4 Method: isDraw()

Checks if the board is completely filled without any empty spaces, indicating a draw.

4.4.5 Method: removeRandomSigns(Player player, int count)

Removes a specified number of the current player's marks randomly from the board, updating the display after changes.

4.4.6 Method: step(int row, int column)

Places the current player's mark on the specified board position if it's empty, then switches to the next player. If the board is full, indicates a draw.

4.5 Player Enum

4.5.1 Components:

X, O, NOBODY

5 Connections between the events and event handlers.

5.0.1 Window Class Event Handler

Places the current player's mark on the specified board position if it's empty, then switches to the next player. If the board is full, indicates a draw.

Event: Clicking on the game board buttons.

Event Handler: In the Window class, the addButton method creates the buttons for the game board. Each button has an ActionListener that listens for a click event. When a button is clicked, it triggers the event handler, which calls the step(i, j) method of the Model class, updating the game state (i.e., placing the current player's mark at the selected position). Then, it updates the board display and checks for a winner or draw.

Event: Clicking the "New Game" button.

Event Handler: The newGameButton has an ActionListener that triggers a new game when clicked. The event handler calls the newGame() method, which creates a new Window and disposes of the current one.

5.0.2 MainWindow Class Event Handler

Event: Event: Clicking on a button for selecting a grid size (e.g., 6x6, 10x10, 14x14).

Event Handler: In the MainWindow class, the grid size buttons (small, medium, big) each have an ActionListener. When a button is clicked, the event handler triggers the creation of a new Window with the selected grid size (by calling new Window(size, MainWindow.this)), which in turn initializes a new game with the specified size.

6 Testing

Test Cases

1. **Click the Same Button**
    - **As a:** creator
    - **I want to:** prevent a player from clicking the same button twice
    - **Given:** the game model is initialized
    - **When:** the player clicks the same button twice
    - **Then:** the value of the button does not change, and no action is performed.
2. **Deletes Three Consecutives**
    - **As a:** creator
    - **I want to:** delete three consecutive signs after placing three in a row
    - **Given:** the game board has three consecutive signs from player X
    - **When:** the player steps on the winning position
    - **Then:** the game removes the player's pieces and triggers the game logic.
3. **Deletes Four Consecutives**
    - **As a:** creator
    - **I want to:** delete four consecutive signs after placing four in a row
    - **Given:** the game board has four consecutive signs from player X
    - **When:** the player steps on the winning position
    - **Then:** the game removes the player's pieces and triggers the game logic.
4. **Test Player Turns**
    - **As a:** creator
    - **I want to:** ensure that turns alternate between players
    - **Given:** the game model starts with player X
    - **When:** player X takes a turn and then player O takes a turn
    - **Then:** the player alternates correctly.
5. **Test Board Initialization**
    - **As a:** creator
    - **I want to:** ensure the board is initialized correctly
    - **Given:** the game model is initialized with a 6x6 board
    - **When:** I inspect the board
    - **Then:** all positions are empty (set to `NOBODY`).
6. **Test Winner Detection**

- **As a:** creator
- **I want to:** detect if a winner is correctly identified
- **Given:** player X places five consecutive signs in a row
- **When:** player X completes five consecutive signs
- **Then:** the game detects player X as the winner.

7. **Test Draw Detection**
   - **As a:** creator
   - **I want to:** detect if the game results in a draw
   - **Given:** the board is completely filled without a winner
   - **When:** the game ends
   - **Then:** the game correctly identifies the draw.

8. **Test Player Alternates**
   - **As a:** creator
   - **I want to:** ensure players alternate between moves
   - **Given:** the game model starts with player X
   - **When:** player X and player O take turns
   - **Then:** the players alternate correctly between turns.

9. **Test Different Board Sizes**
   - **As a:** player
   - **I want to:** play on different sized boards
   - **Given:** different board sizes (e.g., 6x6, 10x10, 14x14)
   - **When:** I select a board size
   - **Then:** the board initializes correctly with the given size.

10. **Test Empty Board**
    - **As a:** creator
    - **I want to:** ensure the board is empty when initialized
    - **Given:** a 6x6 board is initialized
    - **When:** I inspect the board
    - **Then:** all positions are empty (set to `NOBODY`).

11. **Test Full Board No Winner**
    - **As a:** creator
    - **I want to:** detect if the game ends with no winner on a full board
    - **Given:** the board is filled without five consecutive signs
    - **When:** I inspect the board after all moves
    - **Then:** the game identifies no winner and ends in a draw.

12. **Test Immediate Victory**
    - **As a:** creator
    - **I want to:** test if a player can win immediately
    - **Given:** player X makes the first move
    - **When:** I check for a winner after the first move
    - **Then:** the game identifies that no winner is present.
13. **Test Diagonal Win**
    - **As a:** creator
    - **I want to:** check if the game detects a diagonal win
    - **Given:** player X places five consecutive signs diagonally
    - **When:** I check for a winner
    - **Then:** the game correctly detects player X as the winner.
14. **Test Vertical Win**
    - **As a:** creator
    - **I want to:** check if the game detects a vertical win
    - **Given:** player X places five consecutive signs vertically
    - **When:** I check for a winner
    - **Then:** the game correctly detects player X as the winner.