**Tron**

Written and Implemented By

Mohammed Efaz (ZTFK53)

Presented To: Dr Hoque A H M Sajedul

3rd Semester | 3rd December 2023

Faculty of Informatics

Programming Technology | IP-18fPROGTEG

Eötvös Loránd University

Contents

[**Task** 3](#_Toc149768807)

[**Description of the task** 3](#_Toc149768808)

[**UML Diagram** 3](#_Toc149768809)

[**Description of the methods** 4](#_Toc149768810)

[**Event-Handler Connections** 4](#_Toc149768811)

[**Test** 4](#_Toc149768812)

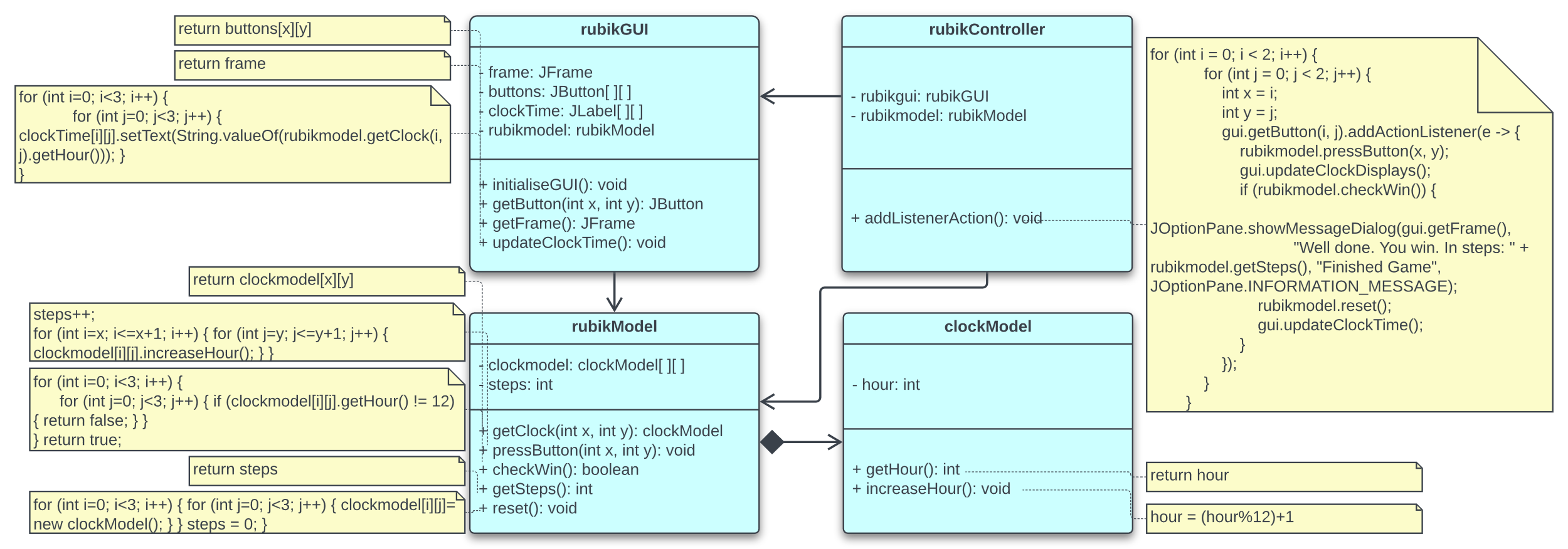
**Task**

Create a game, with we can play the light-motorcycle battle (known from the Tron movie) in a top view. Two players play against each other with two motors, where each motor leaves a light trace behind of itself on the display. The motor goes in each second toward the direction, that the player has set recently. The first player can use the WASD keyboard buttons, while the second one can use the cursor buttons for steering. A player loses if its motor goes to the boundary of the game level, or it goes to the light trace of the other player. Ask the name of the players before the game starts and let them choose the colour their light traces. Increase the counter of the winner by one in the database at the end of the game. If the player does not exist in the database yet, then insert a record for him. Create a menu item, which displays a high score table of the players for the 10 best scores. Also, create a menu item which restarts the game.

**Description of the task**

The classes are created (apart from the main): rubikGUI, rubikModel, rubikController, clockModel. This follows the MVC (Model-View-Controller) design pattern which means that the code is reusable, maintainable, and easily editable (if need be, in the future).

**UML Diagram**

****

**rubikGUI has an association with rubikModel, meaning it interacts with rubikModel but doesn't manage its life.**

**rubikModel has a *composition* relationship with clockModel. This means rubikModel is made up of clockModel objects; if rubikModel is destroyed, so are the contained clockModel objects.**

**rubikController has associations with both rubikGUI and rubikModel, coordinating their interactions without owning them.**

**Description of the methods**

**rubikGUI (GUI)**

* **initialiseGUI()**: Initialises the main GUI component of the software. It includes the clock times, the buttons, and the overall frame/layout.
* **getButton(int x, int y)**: Brings out the button at the given numbers in the Euclidean plane.
* **getFrame()**: Brings out the main frame of the GUI component.
* **updateClockTime()**: Updates the clock times for showing the new hours.

**rubikModel (Model)**

* **getClock(int x, int y)**: Brings out the clock time at the given numbers in the Euclidean plane.
* **pressButton(int x, int y)**: When pressed, updates the adjacent clocks time by adding 1 (hour).
* **checkWin()**: Returns the true if all the clocks time is 12 (hours).
* **getSteps()**: Returns the frequency of the button pressed after which all the clocks time are 12 (hours).
* **reset()**: Resets the game to the beginning stage.

**rubikController (Controller)**

* **addListenerAction()**: Adds the action listeners for buttons, allowing the game, to effectively, run.

**clockModel (Model)**

* **getHour()**: Returns the current time (in hour) on the respective clock.
* **increaseHour()**: Increases the clock’s time (hour) by one, in a 12-hour format.

**Event-Handler Connections**

**pressButton Event**: In the **rubikController** class, the **addListenerAction()** method establishes event handlers for button press events. It uses a nested loop to iterate through the buttons in the **rubikGUI**, and for each button, it makes a lambda expression as the event handler.

**pressButton Handling**: When a button is clicked in the GUI, the accompanying event handler is triggered. The lambda expression takes in the button's position (x, y), and the **rubikModel** is updated by calling the **pressButton()** method.

**Game Finished Check**: After each button press, the event handler checks if the game is completed by calling the **checkWin()** method from the **rubikModel**. If all clocks display "12," a message box is displayed to inform the player that they won.

**Game Reset**: In case of a win, the event handler resets the game by calling the **rubikModel**‘s **reset()** method to start a new game.

**Test**

* Testing whether the close button executes the windows or just hides it in the task manager.
* Testing whether the game outputs the correct count when winning it.
* Testing whether clicking the button updates its adjacent clocks to increase their respective time by 1 (hour).