Rubiks Clock

Written and Implemented

By

Mohammed Efaz (ZTFK53)

3rd Semester | 1st November 2023

Programming Technology

Eötvös Loránd University

Contents

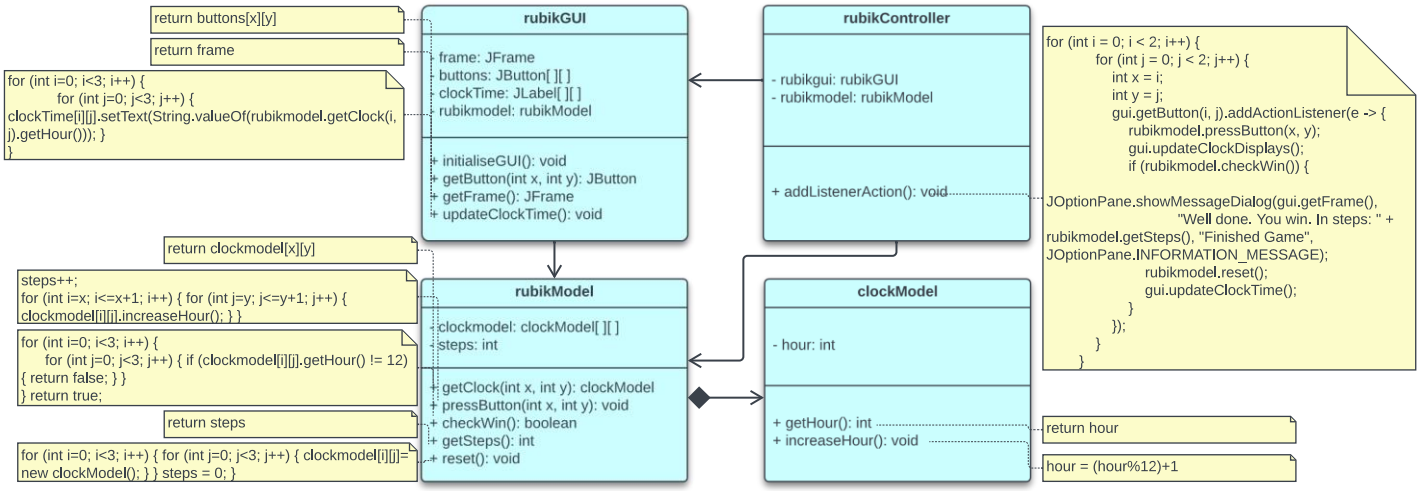
Task

Create a game, which implements the Rubik clock. In this game there are 9 clocks. Each clock can show a time between 1 and 12 (hour only). Clocks are placed in a 3x3 grid, and initially they set randomly. Each four clocks on a corner have a button placed between them, so we have four buttons in total. Pressing a button increase the hour on the four adjacent clocks by one. The player wins, if all the clocks show 12. Implement the game, and let the player restart it. The game should recognize if it is ended, and it must show in a message box how much steps did it take to solve the game. After this, a new game should be started automatically.

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).