Jessica Shayya **2. assignment/10. task** 13th November 2024
UZ47BG
uz47bg@inf.elte.hu
Group 7

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

Class: RubiksClockGame

- **Attributes**:
  - *clockValues*: : *A 3x3 matrix of integers representing the values of the clocks (1 to 12).*
  - *steps: Counter for the number of rotations made*
- **Methods**:
  - + *getClockValues(): Returns the current state of the clocks.*
  - + *rotateClocks(int control): Rotates the clocks in the specified quadrant (control value ranges from 0 to 3).*
  - + *checkWinCondition(): Checks if all clocks are set to 12.*
  - + *getSteps(): Returns the number of steps taken.*
  - + *resetGame(): Resets the game to its initial state.*

Class: RoundButton

- **Attributes**:
  - *label: The text label (the number) displayed inside the circular button.*
- **Methods**:
  - + *paintComponent(Graphics g): Custom rendering method to display the button as a circle.*
  - + *paintBorder(Graphics g): Draws the circular border around the button.*
  - + *getPreferredSize(): Ensures the button remains circular.*
  - + *contains(int x, int y): Checks if a point (x, y) lies inside the circular button.*

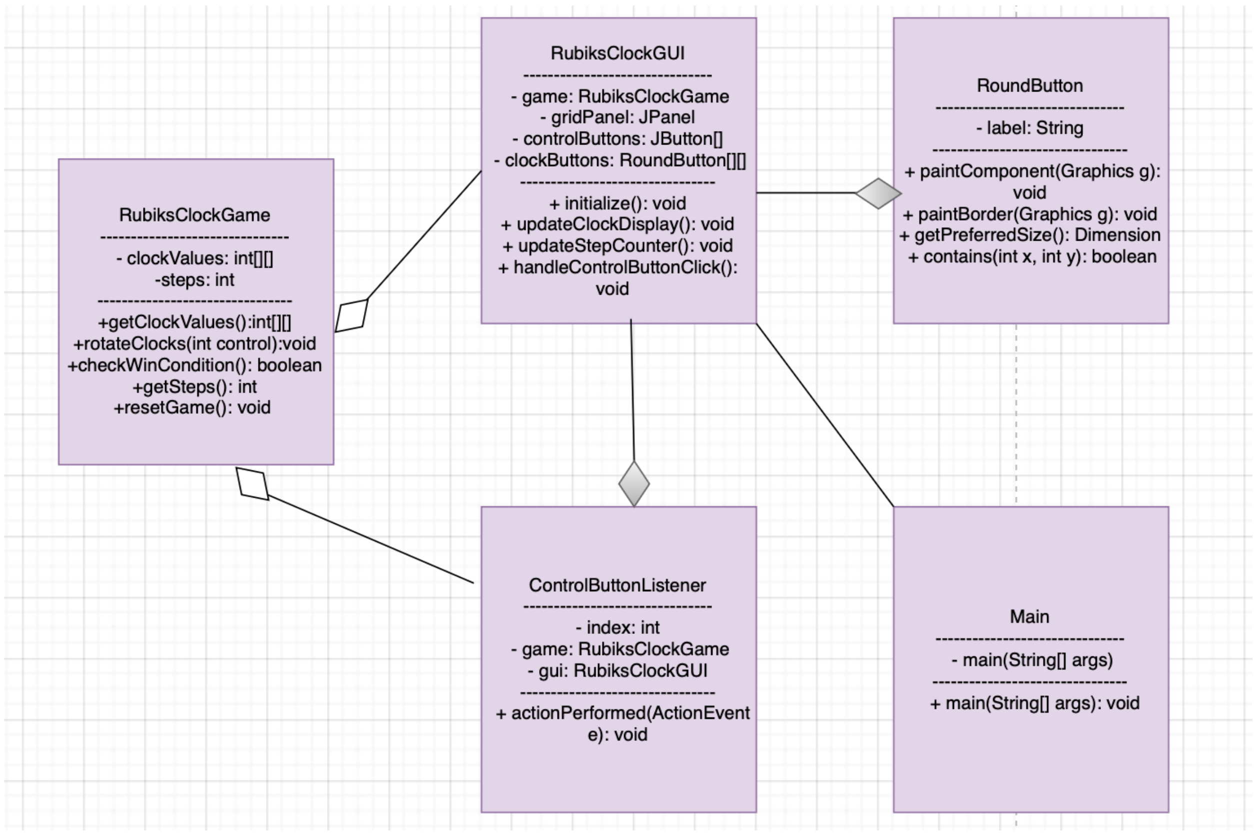
Class: RubiksClockGUI

- **Attributes**:
  - – *game: The RubiksClockGame instance that holds the game logic.*
  - – *gridPanel: A panel that holds the visual grid of clocks.*
  - – *controlButtons: Array of buttons for rotating the quadrants.*
  - – *clockButtons: 2D array of RoundButton instances representing the clocks.*
- **Methods**:
  - + *initialize(): Initializes the game grid and control buttons.*
  - + *updateClockDisplay(): Updates the clock display based on the current game state.*
  - + *updateStepCounter(): Updates the step counter display.*
  - + *handleControlButtonClick(): Handles the logic when a control button is clicked (rotating the quadrant).*

Class: ControlButtonListener

- **Attributes**:
  - – *index: The index of the control button (which quadrant to rotate).*
  - – *game: A reference to the RubiksClockGame instance to interact with the game logic*
  - – *gui: A reference to the RubiksClockGUI to update the display.*
- **Methods**:
  - + *actionPerformed(ActionEvent e): Handles the button click events and rotates the corresponding quadrant*

Class Diagram


RubiksClockGUI
- game: RubiksClockGame
- gridPanel: JPanel
- controlButtons: JButton[]
- clockButtons: RoundButton[][]
+ initialize(): void
+ updateClockDisplay(): void
+ updateStepCounter(): void
+ handleControlButtonClick(): void
RoundButton
- label: String
+ paintComponent(Graphics g): void
+ paintBorder(Graphics g): void
+ getPreferredSize(): Dimension
+ contains(int x, int y): boolean
RubiksClockGame
- clockValues: int[][]
-steps: int
+getClockValues():int[][]
+rotateClocks(int control):void
+checkWinCondition(): boolean
+getSteps(): int
+resetGame(): void
ControlButtonListener
- index: int
- game: RubiksClockGame
- gui: RubiksClockGUI
+ actionPerformed(ActionEvent e): void
Main
- main(String[] args)
+ main(String[] args): void

Edge Case Testing

1- testInitialization(): Verifies that all clock values are initialized within the valid range (1 to 12).

- AS A user
- I WANT TO ensure initial clock values are between 1 and 12
- GIVEN a newly initialized game
- WHEN I check each clock value
- THEN each should be within the valid range (1 to 12)

2- testRotateTopLeftControl(): Ensures that rotating the top-left control correctly updates the clock values within that quadrant.

- AS A user
- I WANT TO rotate the clocks in the top-left quadrant
- GIVEN a predefined set of clock values
- WHEN I perform a rotation on the top-left control
- THEN only the specified clocks should increment by 1

3- testRotateTopRightControl():Verifies that rotating the top-right control affects only the top-right quadrant clocks.

- AS A user
- I WANT TO rotate the clocks in the top-right quadrant
- GIVEN a predefined set of clock values
- WHEN I perform a rotation on the top-right control
- THEN only the specified clocks should increment by

4- testRotateBottomLeftControl():Checks that rotating the bottom-left control only impacts the bottom-left quadrant clocks.

- AS A user
- I WANT TO rotate the clocks in the bottom-left quadrant
- GIVEN a predefined set of clock values
- WHEN I perform a rotation on the bottom-left control
- THEN only the specified clocks should increment by 1

5- testRotateBottomRightControl():Verifies that rotating the bottom-right control updates only the bottom-right quadrant clocks.

- AS A user
- I WANT TO rotate the clocks in the bottom-right quadrant
- GIVEN a predefined set of clock values
- WHEN I perform a rotation on the bottom-right control
- THEN only the specified clocks should increment by 1

6- testStepCounter():Confirms that the step counter increments each time a control is rotated.

- AS A user
- I WANT TO track the number of steps taken
- GIVEN the initial step count
- WHEN I rotate a control
- THEN the step counter should increase by 1

7- testWrapAroundAt12():Ensures that clock values wrap around from 12 back to 1 when rotated.

- AS A user
- I WANT TO observe correct wrap-around behavior for clock values
- GIVEN a clock set to 12
- WHEN I rotate the control containing that clock
- THEN the clock value should wrap around to 1

8- testWinConditionAll12s():Verifies that the game detects a win when all clock values are set to 12.

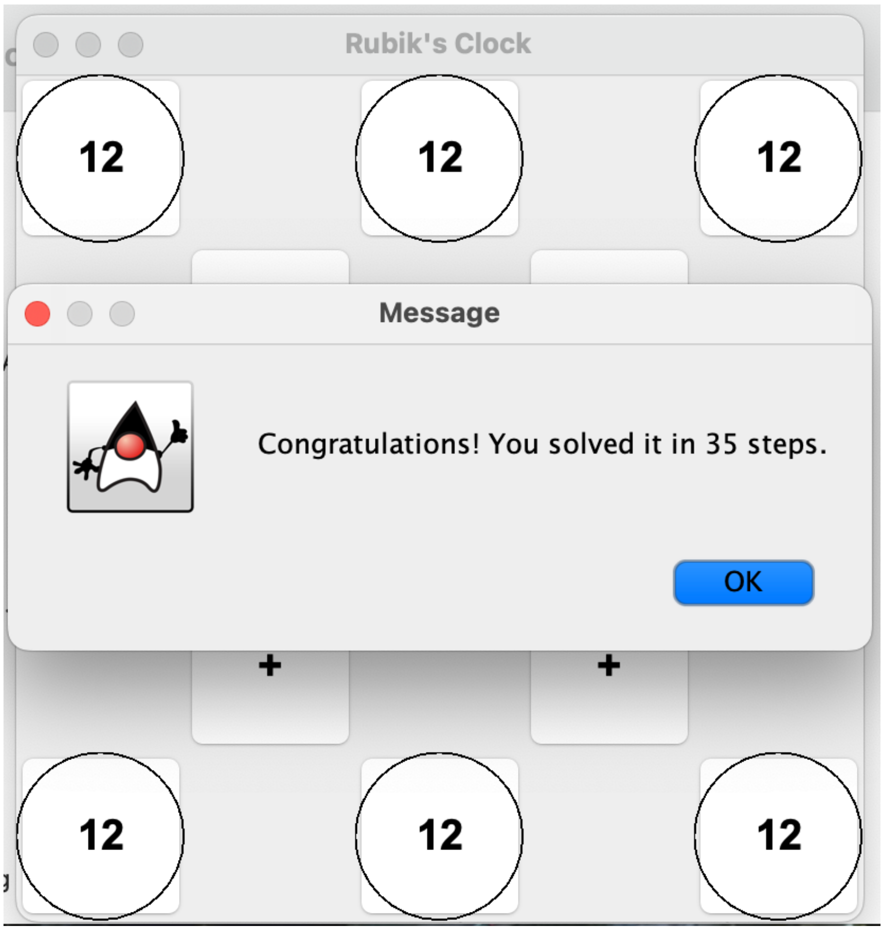
- AS A user
- I WANT TO achieve a win condition
- GIVEN all clocks set to 12
- WHEN I check the win condition
- THEN the game should recognize it as a win

9- testNonWinCondition():Checks that the game does not detect a win when any clock value is not 12.

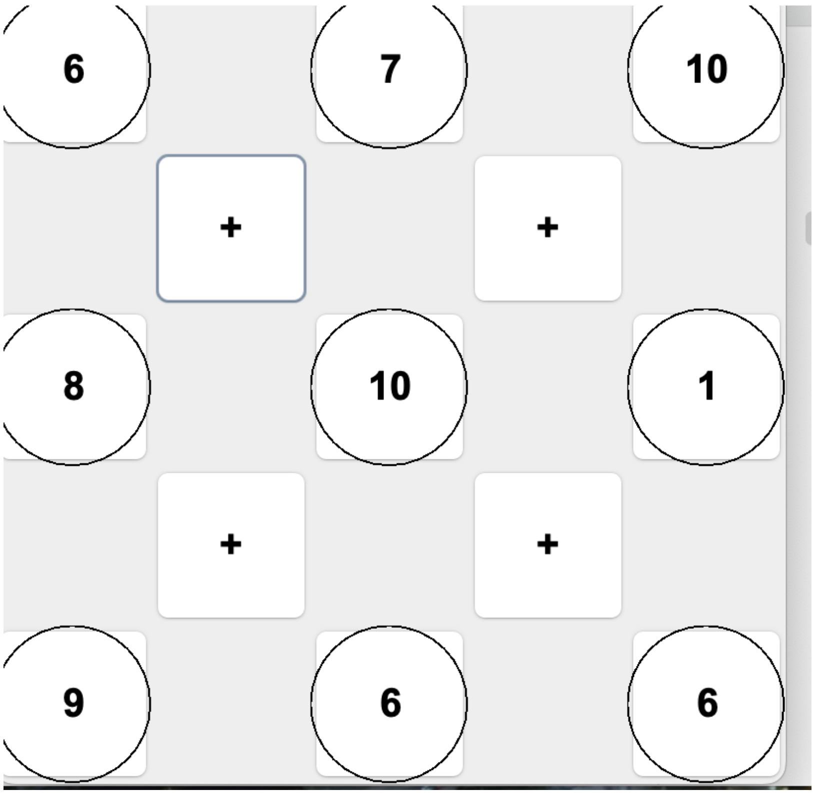
- AS A user
- I WANT TO verify that a non-winning state is not falsely recognized
- GIVEN one clock set to a non-12 value
- WHEN I check the win condition
- THEN the game should not recognize it as a win

10- testResetGame():Ensures that resetting the game sets each clock to a new random value between 1 and 12 and resets the step counter.

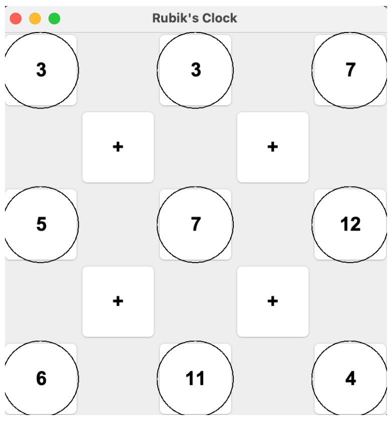
- AS A user
- I WANT TO restart the game with initial conditions
- GIVEN a game in progress
- WHEN I reset the game
- THEN each clock value should be within the valid range, and the step counter should be 0



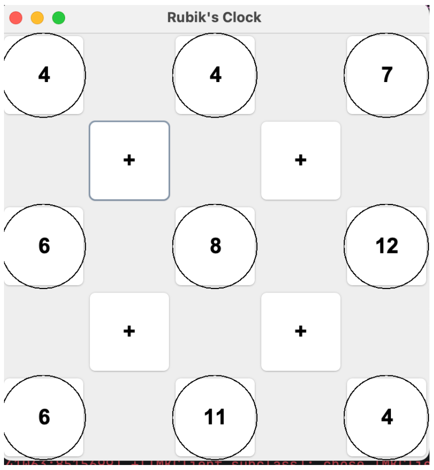
Congratulations! You solved it in 35 steps.



Rotating top left:

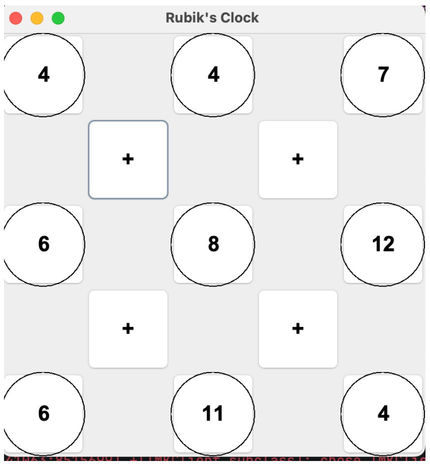


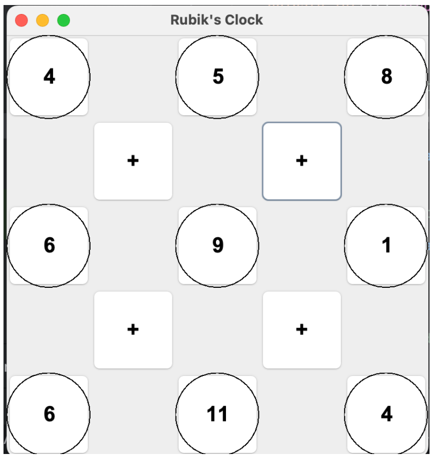
Before



After

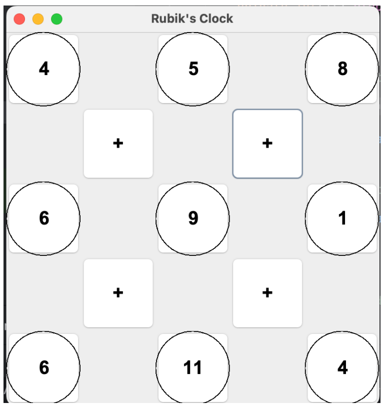
Rotating top right:



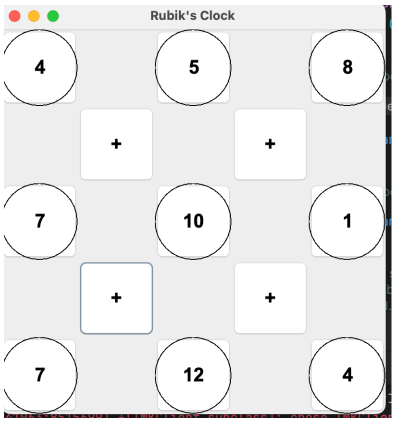


After

Rotating bottom left:

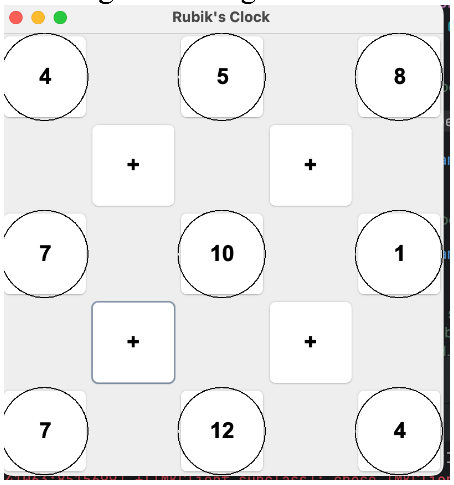


Before

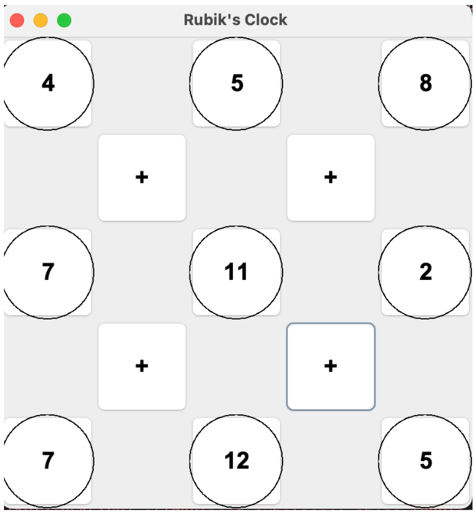


After

Rotating bottom right



Before



After