Report On

Car Race Game

Submitted in partial fulfillment of the requirements of the Course project in
Semester III of Second Year Artificial Intelligence and Data Science

by
Mitanksh Gosalia (Roll No. 13)
Nitish Jha (Roll No. 18)
Kiran Dhuri (Roll No. 07)

Supervisor
Prof. Sneha M. Yadav

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

(2023-24)

Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science

CERTIFICATE

This is to certify that the project entitled “Car Race Game ” is a bonafide work of " Mitanksh Gosalia (Roll No.13), Nitish Jha (Roll No. 18), Kiran Dhuri (Roll No. 07)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester III of Second Year Artificial Intelligence and Data Science engineering.

Supervisor

Prof. Name Surname

Dr. Tatwadarshi P. N.
Head of Department

Table of Contents

Pg. No

Abstract

The "CarRaceGame" is a Java-based graphical application that simulates a simple car racing game. The game is implemented using the Java AWT (Abstract Window Toolkit) library and provides a basic user interface for controlling a car's horizontal movement using the keyboard's left and right arrow keys. The game window consists of a green background and a black rectangular car that can be moved left and right within the window boundaries. The primary goal of this game is to navigate the car and avoid any hypothetical obstacles.

The game utilizes Java's event-driven programming model to respond to keyboard input, where the KeyListener interface is implemented to handle key presses and releases. A separate thread is used to continuously update the car's position, creating a smooth animation effect. The car's movement is controlled by setting the "left" and "right" boolean variables, which are modified based on key input and used in the game's update loop.

The game provides a simple and interactive way for users to experience a basic car racing scenario. Players can control the car's movement to navigate through the virtual road. The game loop ensures real-time updates and smooth animation of the car's movement. While this is a minimalistic example of a car racing game, it serves as a foundation for more advanced and feature-rich game development in Java.

Problem Statement

Develop a Java-based car racing game using the AWT library, where players control a car's horizontal movement using the left and right arrow keys. The game features a green background representing the track, a black rectangular car, and implements a real-time game loop for animation. Players aim to navigate the car through the track while avoiding obstacles (not explicitly implemented in the provided code), offering an engaging gaming experience.

Module Description:

Module Description:

1. **User Interface Module:**
   - This module is responsible for creating the game window using AWT.
   - It sets up the initial graphical environment, including the green background and the car's visual representation.
   - It displays the game title and any instructions or controls for the player.

2. **Input Handling Module:**
   - This module implements the KeyListener interface to capture user keyboard input.

- It responds to left and right arrow key presses and releases, updating the "left" and "right" boolean variables to control the car's movement.

3. **Game Loop Module:**
   - The game loop is implemented using a separate thread (Runnable) for continuous updates.
   - This module manages the car's movement and animation, ensuring a smooth and responsive gaming experience.
   - It contains logic to check the "left" and "right" variables to move the car accordingly and repaint the graphics.

4. **Graphics Rendering Module:**
   - This module handles the rendering of the game graphics.
   - It paints the green background and the black rectangular car, updating their positions as directed by the game loop.
   - It ensures that the visuals are continuously refreshed for the player.

5. **Collision Detection Module (optional):**
   - If obstacles or boundaries are to be introduced in the game, a collision detection module can be added.
   - This module checks for collisions between the car and obstacles and takes appropriate action when a collision occurs, such as ending the game or deducting points.

6. **Error Handling and Termination Module:**
   - This module handles exceptions and errors that may occur during the game's execution.
   - It ensures a graceful termination of the game when the player closes the window, releasing any allocated resources.

7. **Documentation and User Interface Enhancement Module:**
   - This module includes code comments for clarity and maintainability.
   - It provides documentation to explain the code structure, components, and how to run the game.
   - It focuses on improving the user interface for an intuitive and engaging user experience, such as adding a score display or winning conditions (optional).

Brief Description of Software and Hardware Used and Its Programming for theTic Tac Toe with GUI using java:

Software Used:
Brief Description of Software, Hardware, and Programming:

Software:
1. **Java Development Kit (JDK):** The game is developed using Java, and the JDK provides the necessary tools for Java programming, including the Java compiler and runtime environment.

2. **Integrated Development Environment (IDE):** An IDE like Eclipse, IntelliJ IDEA, or NetBeans can be used to write, debug, and run the Java code. IDEs provide features like code editing, debugging tools, and project management.

3. **AWT (Abstract Window Toolkit):** AWT is a Java library used to create graphical user interfaces. It provides classes for building windows, handling events, and drawing graphics on the screen.

Hardware:
1. **Computer:** A standard computer system with sufficient processing power and memory to run the Java development environment and execute the game. The hardware requirements are typically minimal for Java-based applications.

2. **Keyboard:** A keyboard is required to control the car's movement within the game. Players use the left and right arrow keys for steering.

Programming:

The programming for the car racing game involves the following key components:

1. **Class Structure:** The game is organized into a Java class named "CarRaceGame" that extends the AWT Frame class. This class contains the main game logic, including GUI setup, event handling, game loop, and graphics rendering.

2. **User Interface:** AWT is used to create the game window with a green background and draw the car. The `paint` method is overridden to define the graphics rendering process.

3. **Event Handling:** The `KeyListener` interface is implemented to capture user keyboard input. Key events (key presses and releases) are handled in the `keyPressed` and `keyReleased` methods.

4. **Game Loop:** A separate thread is created to run the game loop. The `run` method is responsible for continuously updating the car's position, checking user input, and refreshing the graphics.

5. **Graphics Rendering:** The game uses AWT graphics to render the green background and the black rectangular car. The `repaint` method is used to trigger graphic updates.

6. **Optional Collision Detection:** If obstacles or boundaries are introduced, collision detection logic can be added to check for collisions between the car and obstacles.

7. **Error Handling:** Appropriate error handling, such as exception handling, is implemented to ensure the game operates smoothly and terminates gracefully.

8. **Documentation:** Code comments are added for code clarity, and comprehensive documentation describes the program structure, components, and instructions for running the game.

The combination of software (Java, AWT), hardware (a standard computer), and programming techniques results in a simple yet interactive car racing game that can be run and enjoyed on a Java-enabled system with a keyboard for control.

CODE:

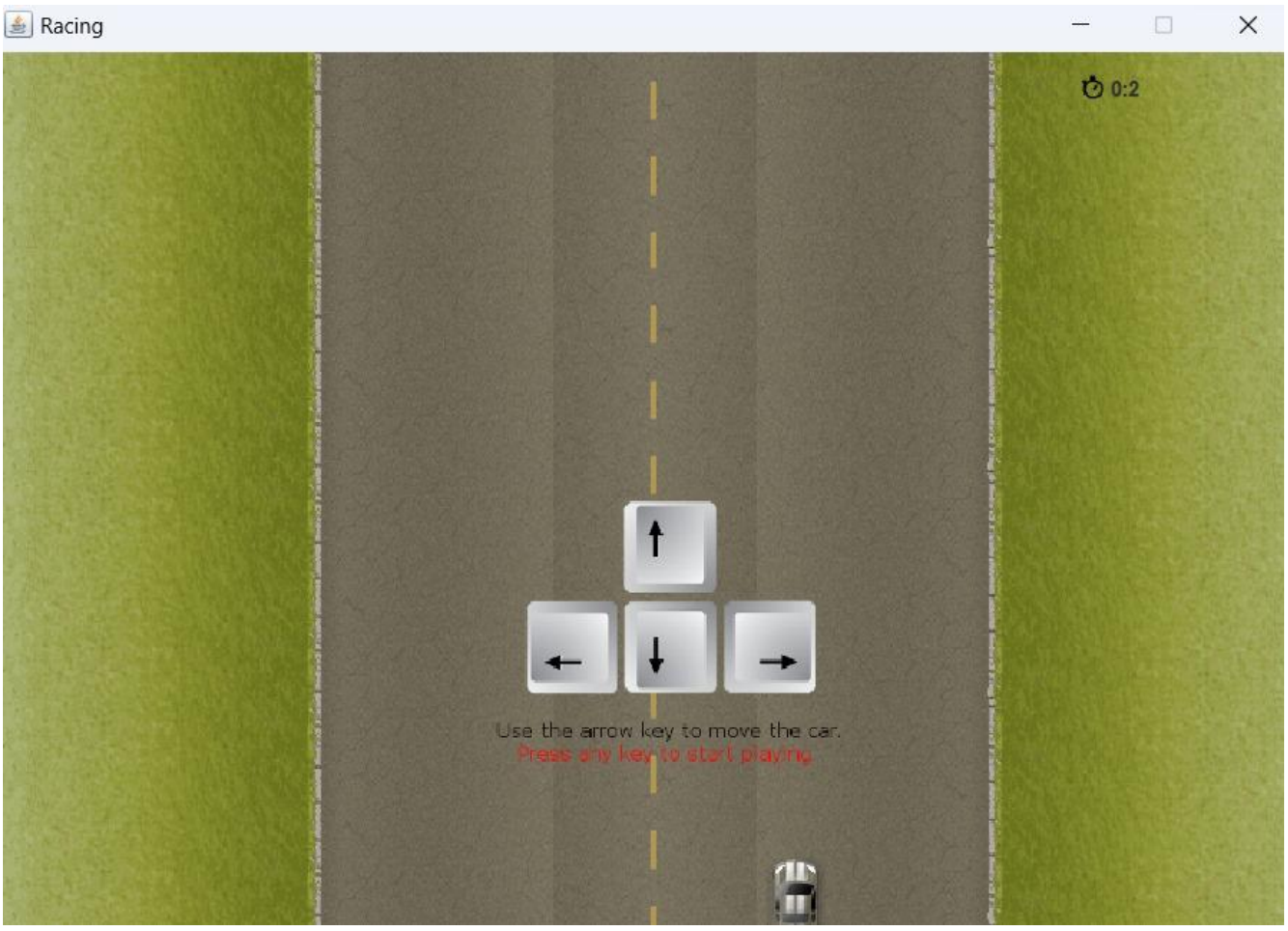
```
import java.awt.*;
```

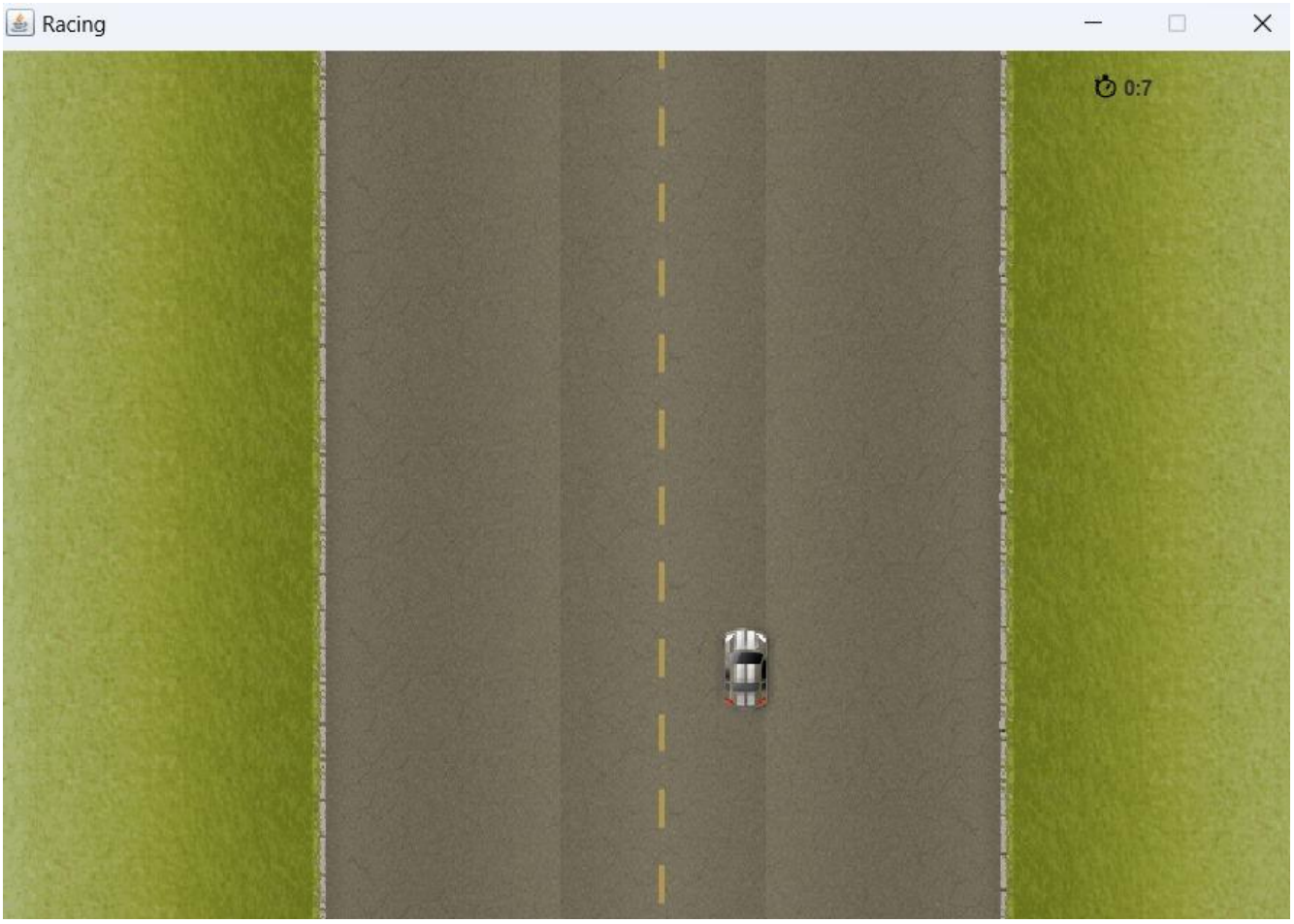
```
import java.awt.event.*;

public class CarRaceGame extends Frame implements KeyListener, Runnable {
    private int carX = 200;
    private int carY = 400;
    private boolean left, right;

    public CarRaceGame() {
        setSize(400, 500);
        setTitle("Car Race Game");
        setVisible(true);
        addKeyListener(this);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        });
        Thread t = new Thread(this);
        t.start();
    }

    public void paint(Graphics g) {
        g.setColor(Color.GREEN);
        g.fillRect(0, 0, getWidth(), getHeight());

        g.setColor(Color.BLACK);
        g.fillRect(carX, carY, 50, 30);
    }

    public void keyPressed(KeyEvent ke) {
        if (ke.getKeyCode() == KeyEvent.VK_LEFT) {
            left = true;
        }
```

```
    if (ke.getKeyCode() == KeyEvent.VK_RIGHT) {
        right = true;
    }
}

public void keyReleased(KeyEvent ke) {
    if (ke.getKeyCode() == KeyEvent.VK_LEFT) {
        left = false;
    }
    if (ke.getKeyCode() == KeyEvent.VK_RIGHT) {
        right = false;
    }
}

public void keyTyped(KeyEvent ke) {}

public void run() {
    while (true) {
        if (left && carX > 0) {
            carX -= 5;
        }
        if (right && carX < getWidth() - 50) {
            carX += 5;
        }

        repaint();

        try {
            Thread.sleep(20);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
    }

    public static void main(String[] args) {
        new CarRaceGame();
    }
}
```

Results and Conclusion:





Conclusion:

In conclusion, the development of the Java-based car racing game using the AWT library showcases the fundamental principles of event-driven programming, graphic rendering, and user interface design. This project provides an engaging and interactive gaming experience, allowing players to control a car's movement within a visually appealing virtual racing track. The use of Java and AWT, combined with a real-time game loop, enables a responsive and enjoyable gaming experience. Further enhancements, such as obstacle collisions and additional features, can be integrated to make this project a foundation for more complex game development. Overall, this project illustrates the versatility and potential of Java in creating interactive graphical applications.