Didar Baltayev(B1905.090085)

Computer Programming Project

Catch the Drop

Introduction

This is a simple Java Swing game where a droplet falls from the top of the screen. The player needs to click on the droplet before it reaches the bottom. If the player successfully clicks the droplet, the score increases, and the speed of the droplet also increases. If the droplet reaches the bottom without being clicked, a "game over" image is displayed.

Code

```
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.IOException;
public class GameWindow extends JFrame {
    private static GameWindow game_window;
    private static long last_frame_time;
    private static Image background;
    private static Image game_over;
    private static Image drop;
    private static float drop_left = 200;
    private static float drop_top = -100;
    private static float drop_v = 200;
    private static int score;
```

```
public static void main(String[] args) throws IOException {
        background = ImageIO.read(GameWindow.class.getResourceAsStream("background.png"));
        drop = ImageIO.read(GameWindow.class.getResourceAsStream("drop.png"));
        game_over = ImageIO.read(GameWindow.class.getResourceAsStream("game_over.png"));
        game_window = new GameWindow ();
        game_window.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        game_window.setLocation(200, 100);
        game_window.setSize(906, 478);
        game_window.setResizable(false);
        last_frame_time = System.nanoTime();
        GameField game_field = new GameField();
        game_field.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                int x = e.getX();
                int y = e.getY();
                float drop_right = drop_left + drop.getWidth(null);
                float drop_botton = drop_top + drop.getHeight(null);
                boolean is_drop = x >= drop_left && x <= drop_right && y >= drop_top && y <= drop_botton;
                if (is_drop){
                    drop_top = -100;
                    drop_left = (int) (Math.random() * (game_field.getWidth() - drop.getWidth(null)));
                    drop_v = drop_v + 20;
                    score++;
                    game_window.setTitle("Score: " + score);

                }
            }
        });
        game_window.add(game_field);
        game_window.setVisible(true);
    }
    private static void onRepaint(Graphics g){
        long current_time = System.nanoTime();
        float delta_time = (current_time - last_frame_time) * 0.000000001f;
        last_frame_time = current_time;


        drop_top = drop_top + drop_v * delta_time;
        g.drawImage(background, 0, 0, null);
        g.drawImage(drop, (int) drop_left, (int) drop_top, null);
        if (drop_top > game_window.getHeight()){
            g.drawImage(game_over, 280, 120, null);
        }

    }
    private static class GameField extends JPanel{

        @Override
        protected void paintComponent (Graphics g){
            super.paintComponent(g);
            onRepaint(g);
            repaint();
        }
    }
}
```

Methodologies

- javax.imageio.ImageIO: Used to read images.
- **javax.swing***: Imports Swing components like **JFrame** and **JPanel** for the graphical user interface.
- **java.awt.***: Imports AWT components for graphics operations.
- **java.awt.event.MouseAdapter**, **java.awt.event.MouseEvent**: Used to capture and respond to mouse events.
- **java.io.IOException**: Exception handling for input-output operations.

Firstly we will upload necessary libraries.

```
1  import javax.imageio.ImageIO;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6  import java.io.IOException;
   6 usages
```

This class extends Jframe meaning it represents a window in which the game runs.

```
public class GameWindow extends JFrame {
    9 usages
    private static GameWindow game_window;
    3 usages
    private static long last_frame_time;
    2 usages
    private static Image background;
    2 usages
    private static Image game_over;
    5 usages
    private static Image drop;
    4 usages
    private static float drop_left = 200;
    7 usages
    private static float drop_top = -100;
    3 usages
    private static float drop_v = 200;
    2 usages
    private static int score;
```

These variables maintain the game's state:

- **game_window**: Reference to the main game window.
- **last_frame_time**: Used for calculating the time since the last frame was painted, for physics calculations.
- **background**, **game_over**, **drop**: Images for the game's background, game-over screen, and the falling droplet respectively.
- **drop_left**, **drop_top**: Coordinates of the droplet on the beginning.
- **drop_v**: Velocity of the droplet.
- **score**: Player's score.

This is the entry point of the application:

```
no usages
18  public static void main(String[] args) throws IOException {
19      background = ImageIO.read(GameWindow.class.getResourceAsStream( name: "background.png"));
20      drop = ImageIO.read(GameWindow.class.getResourceAsStream( name: "drop.png"));
21      game_over = ImageIO.read(GameWindow.class.getResourceAsStream( name: "game_over.png"));
22      game_window = new GameWindow ();
23      game_window.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
24      game_window.setLocation( x: 200,  y: 100);
25      game_window.setSize( width: 906,  height: 478);
26      game_window.setResizable(false);
27      last_frame_time = System.nanoTime();
28      GameField game_field = new GameField();
29      game_field.addMouseListener(new MouseAdapter() {
```

Here, we use **ImageIO.read()** to load images from the application's resources. These images are stored as static variables for use in rendering.

•The main game window is instantiated and its various properties are set.

•**setDefaultCloseOperation**: Specifies that the application should close when the window's close button is pressed.

•**setLocation**: Initial position of the window when it appears on the screen.

•**setSize**: Sets the dimensions of the game window.

•**setResizable**: Prevents the window from being resized.

```java
game_field.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        float drop_right = drop_left + drop.getWidth( observer: null);
        float drop_botton = drop_top + drop.getHeight( observer: null);
        boolean is_drop = x >= drop_left && x <= drop_right && y >= drop_top && y <= drop_botton;
        if (is_drop){
            drop_top = -100;
            drop_left = (int) (Math.random() * (game_field.getWidth() - drop.getWidth( observer: null)));
            drop_v = drop_v + 20;
            score++;
            game_window.setTitle("Score: " + score);

        }
    }
```

A mouse listener is attached to **game_field**, which detects when the player clicks on it.

It fetches the x and y coordinates of the mouse click event **e**.

•The coordinates of the droplet's right and bottom boundaries are calculated.

•**is_drop** checks whether the mouse click occurred inside the boundaries of the droplet.

•If is_drop true

•The droplet's position is reset to start from the top.

•A random horizontal position is assigned to the droplet.

•The droplet's velocity (**drop_v**) is increased, making it fall faster.

•The player's score is incremented.

•The game window's title is updated to show the current score.

```
1 usage
50 @    private static void onRepaint(Graphics g){
51          long current_time = System.nanoTime();
52          float delta_time = (current_time - last_frame_time) * 0.000000001f;
53          last_frame_time = current_time;
54
55
56          drop_top = drop_top + drop_v * delta_time;
57          g.drawImage(background,  x: 0,  y: 0,  observer: null);
58          g.drawImage(drop, (int) drop_left, (int) drop_top,  observer: null);
59          if (drop_top > game_window.getHeight()){
60              g.drawImage(game_over,  x: 280,  y: 120,  observer: null);
61          }
```

This method is the core logic for updating the game state and rendering elements.

Time calculation: Determines how much time has passed since the last frame. This is used to calculate the droplet's new position.

•The current time is fetched in nanoseconds.

•**delta_time** calculates the time difference between the current frame and the last frame. This is used for smooth animation and physics calculations.

•The **last_frame_time** is then updated to the current time for future calculations.

-The droplet's vertical position is updated based on its velocity

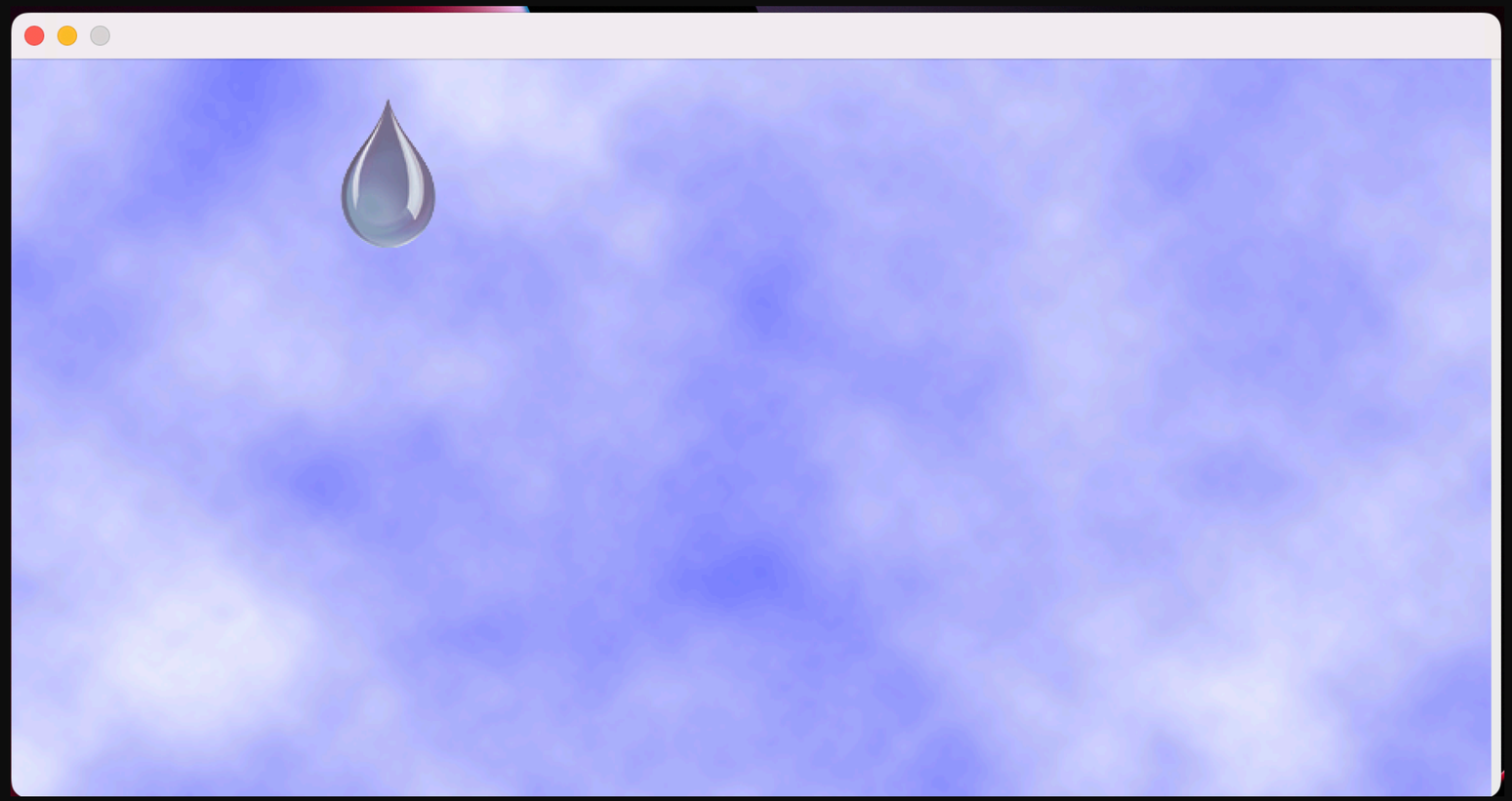
Game Over Check: If the droplet goes below the screen, the game over image is displayed.

Rendering and Animation Loop

- **GameField class**: A custom panel where the game elements (background, droplet, game over image) are drawn.
  - **paintComponent**: This method is called every time Swing wants to draw the panel.
  - **super.paintComponent(g)**: Clears the panel so it's ready for fresh drawing.
  - **onRepaint(g)**: Calls the game logic and drawing method.
  - **repaint()**: Asks Swing to redraw the panel as soon as possible, creating an animation effect.

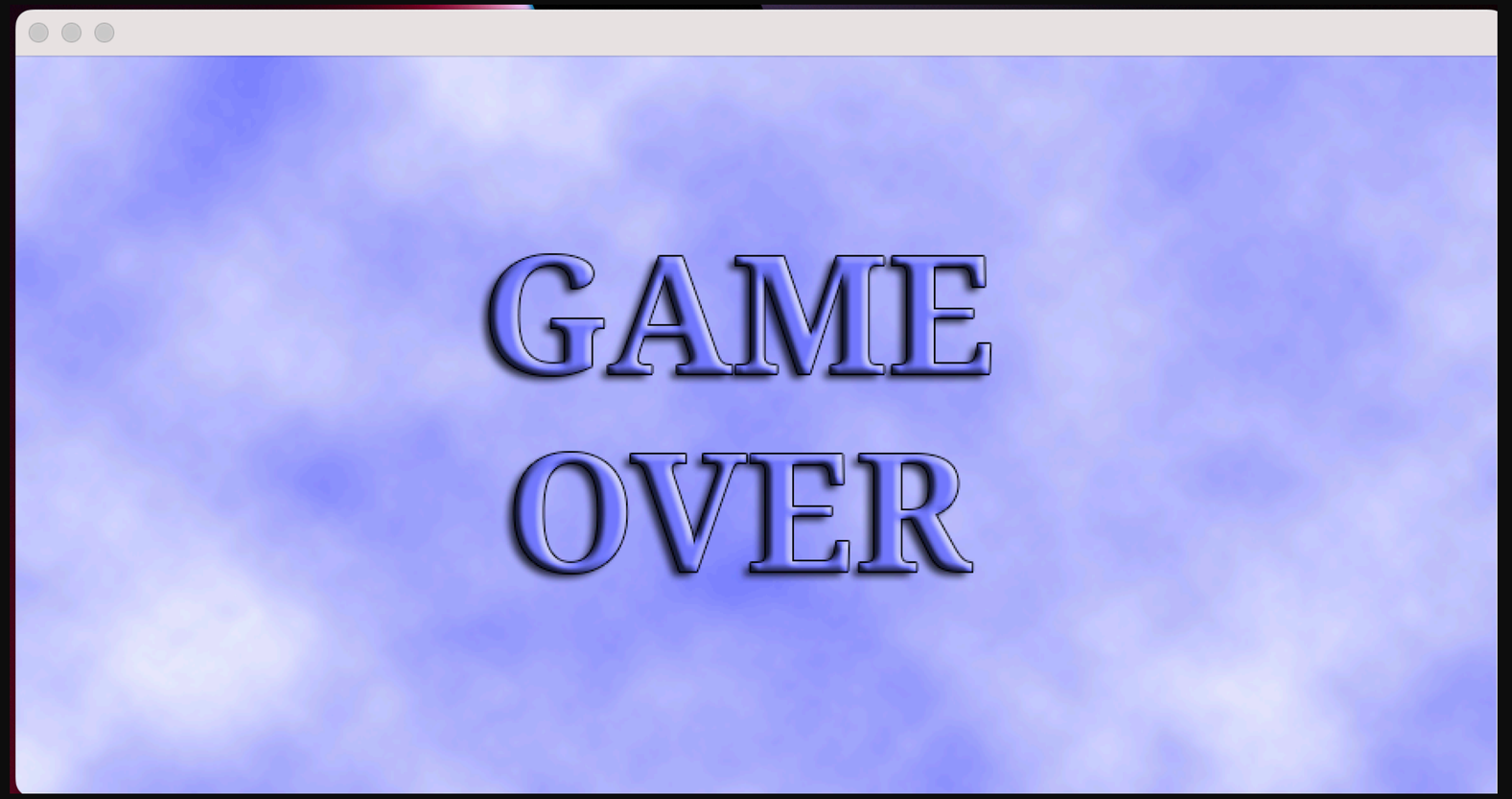
Thanks for attention!

Result



- A window appears with a specified background image.
- The window's size is set to 906x478 pixels, and it's not resizable.
- The window contains a falling "drop" that starts at a random horizontal position at the top of the screen.
- The "drop" falls at an initial velocity, which is represented by the variable **drop_v**.

- If you click on the drop, several things happen:
- The drop's vertical position is reset to the top of the screen (**drop_top = -100**).
- The drop's horizontal position is randomly set within the width of the window.
- The drop's falling speed (**drop_v**) increases by 20 units.
- Your score increases by 1, and the new score is displayed in the title bar of the window.



The game doesn't have an explicit end condition in the code (like closing the window after the "Game Over" message). The drop will keep falling, and if it reaches the bottom, the "Game Over" message will appear, but the game will continue running.