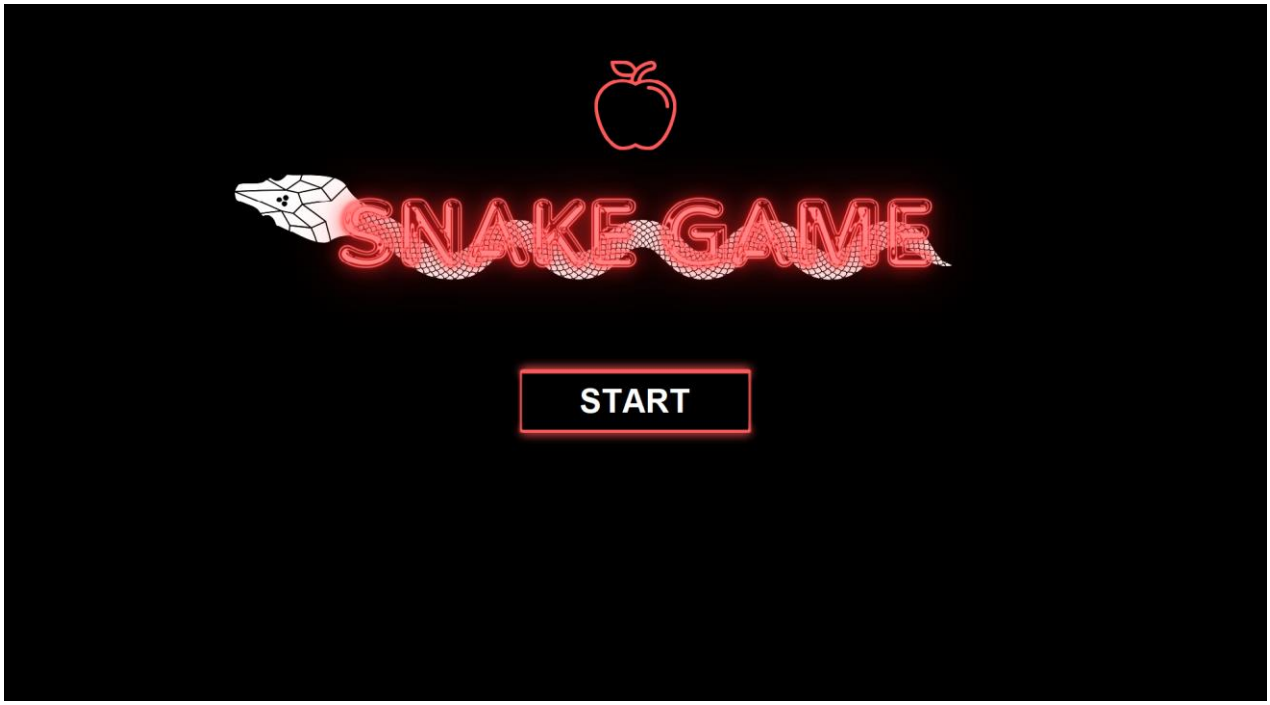
Snake game

Guilherme Cintra

Jhonatan Vitorio

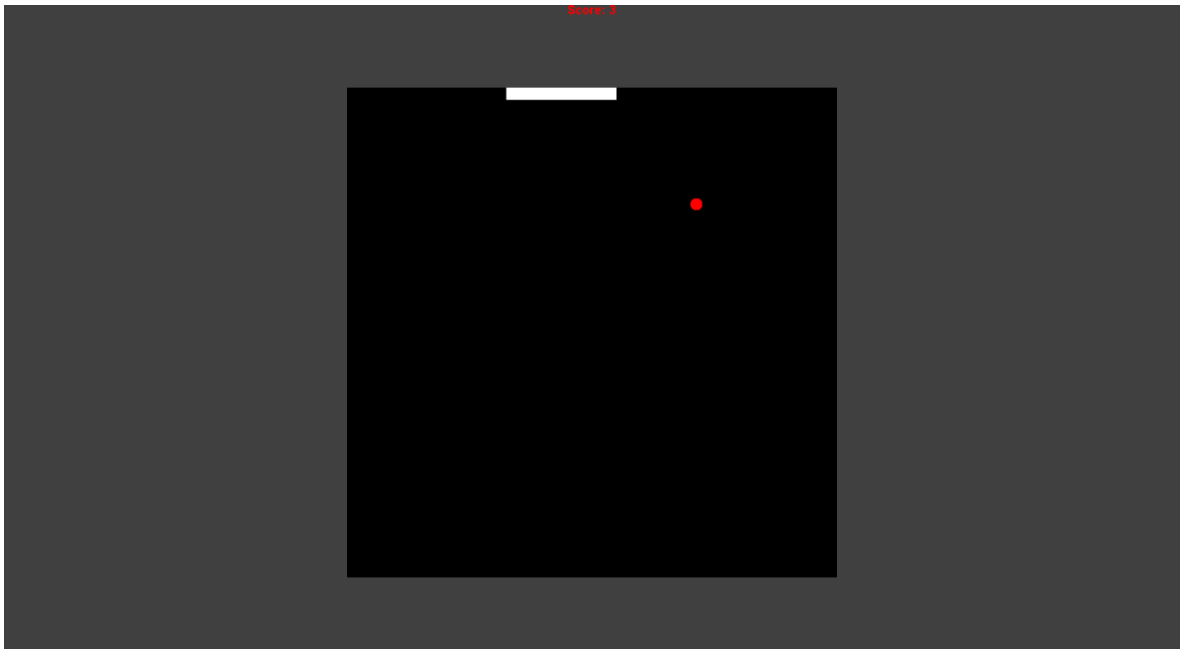
Descriçao:

-Tela inicial:



Quando o jogo é iniciado, uma tela inicial é exibida com um botão “Iniciar” que, quando pressionado, inicia o jogo.

Jogo:



A área de jogo é definida pelo tamanho da tela do usuário.

A cobra começa com um tamanho inicial de seis segmentos e cresce cada vez que come uma maçã.

As maçãs são geradas aleatoriamente na área de jogo.

A cobra é controlada pelo usuário através das teclas de seta ou WASD.

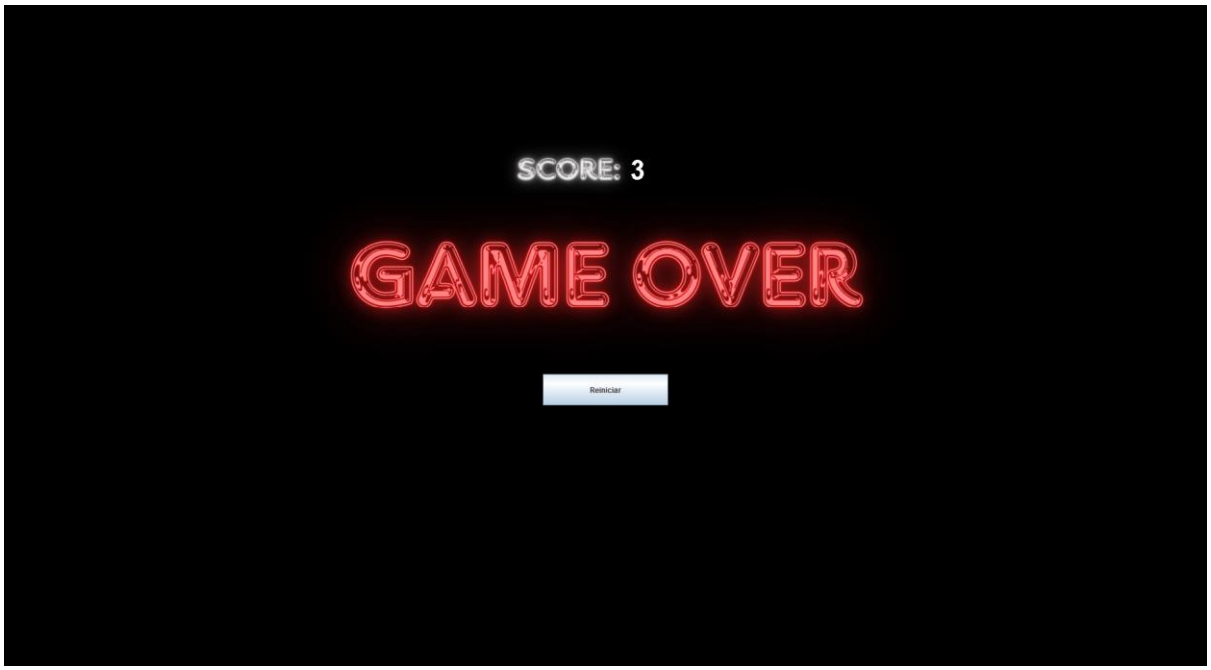
O jogo continua até que a cobra colida com as paredes ou com seu próprio corpo.

-Pontuação:

A pontuação do jogador aumenta cada vez que a cobra come uma maçã.

A pontuação é exibida na tela durante o jogo.

-Game Over:



Quando o jogo termina, uma mensagem de “Game Over” é exibida junto com a pontuação final.

Um botão “Reiniciar” é exibido, permitindo que o jogador comece um novo jogo.

-Detalhes Técnicos:

O jogo é implementado em Java, utilizando a biblioteca Swing para a interface gráfica.

A classe SnakeGame estende JPanel e implementa ActionListener e KeyListener para lidar com eventos de ação e teclado.

O método paintComponent é usado para desenhar os componentes do jogo na tela.

O método actionPerformed é chamado em intervalos regulares pelo Timer, atualizando o estado do jogo.

CÓDIGO FONTE:

```
package snakegame;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SnakeGame extends JPanel implements ActionListener, KeyListener {
    private static final Dimension SCREEN_SIZE = Toolkit.getDefaultToolkit().getScreenSize();
    private static final int SCREEN_WIDTH = (int) SCREEN_SIZE.getWidth();
    private static final int SCREEN_HEIGHT = (int) SCREEN_SIZE.getHeight();
    private static final int UNIT_SIZE = 20;
    private static final int GAME_UNITS = (SCREEN_WIDTH * SCREEN_HEIGHT) / UNIT_SIZE;
    private static final int DELAY = 75;

    private final int[] x = new int[GAME_UNITS];
    private final int[] y = new int[GAME_UNITS];

    private int bodyParts = 6;
    private int appleX;
    private int appleY;
    private char direction = 'R';
    private boolean running = false;
    private boolean isStartScreen = true; // Variável para controlar se está na tela de início
    private Timer timer;
    private int score = 0;

    private JButton startButton;
    private JButton restartButton;

    public SnakeGame() {
        this.setPreferredSize(new Dimension(SCREEN_WIDTH, SCREEN_HEIGHT));
        this.setBackground(Color.BLACK);
        this.setFocusable(true);
        this.addKeyListener(this);

        startButton = new JButton("Iniciar");
        startButton.addActionListener(e -> startGame());
        add(startButton);

        restartButton = new JButton("Reiniciar");
        restartButton.addActionListener(e -> startGame());

        this.add(startButton); // Adiciona o botão de iniciar ao painel inicial
    }

    public void startGame() {
        removeAll(); // Remove todos os componentes do painel
        isStartScreen = false; // Indica que o jogo não está na tela de início
        newApple();
        running = true;
        timer = new Timer(DELAY, this);
        timer.start();
        score = 0;
```

```
    for (int i = 0; i < bodyParts; i++) {
        x[i] = UNIT_SIZE * (bodyParts - i);
        y[i] = UNIT_SIZE;
    }
}

public void drawStartScreen(Graphics g) {
    ImageIcon imageIcon = new ImageIcon("C:\\Users\\valer\\Downloads\\Jogo da cobra (2).png");
    Image image = imageIcon.getImage();
    g.drawImage(image, 0, 0, this.getWidth(), this.getHeight(), null);
}

public void gameOver(Graphics g) {
    g.setColor(Color.RED);
    g.setFont(new Font("Arial", Font.BOLD, 50));
    FontMetrics metrics1 = getFontMetrics(g.getFont());
    g.drawString("Game Over", (SCREEN_WIDTH - metrics1.stringWidth("Game Over")) / 2, SCREEN_HEIGHT / 2 - 50);

    g.setColor(Color.RED);
    g.setFont(new Font("Arial", Font.BOLD, 20));
    FontMetrics metrics2 = getFontMetrics(g.getFont());
    g.drawString("Score: " + score, (SCREEN_WIDTH - metrics2.stringWidth("Score: " + score)) / 2, g.getFont().getSize());
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (isStartScreen) {
        drawStartScreen(g);
    } else if (running) {
        draw(g);
    } else {
        gameOver(g);
    }
}

public void draw(Graphics g) {
    // Desenha a maçã
    g.setColor(Color.RED);
    g.fillOval(appleX, appleY, UNIT_SIZE, UNIT_SIZE);

    // Desenha a cobra
    for (int i = 0; i < bodyParts; i++) {
        if (i == 0) {
            g.setColor(Color.GREEN);
        } else {
            g.setColor(new Color(45, 180, 0));
        }
        g.fillRect(x[i], y[i], UNIT_SIZE, UNIT_SIZE);
    }

    // Desenha a pontuação
    g.setColor(Color.RED);
    g.setFont(new Font("Arial", Font.BOLD, 20));


    FontMetrics metrics = getFontMetrics(g.getFont());
    g.drawString("Score: " + score, (SCREEN_WIDTH - metrics.stringWidth("Score: " + score)) / 2, g.getFont().getSize());
}

public void newApple() {
    appleX = (int) (Math.random() * (SCREEN_WIDTH / UNIT_SIZE)) * UNIT_SIZE;
    appleY = (int) (Math.random() * (SCREEN_HEIGHT / UNIT_SIZE)) * UNIT_SIZE;
}

public void move() {
    for (int i = bodyParts; i > 0; i--) {
        x[i] = x[i - 1];
        y[i] = y[i - 1];
    }

    switch (direction) {
        case 'U':
            y[0] = y[0] - UNIT_SIZE;
            break;
        case 'D':
            y[0] = y[0] + UNIT_SIZE;
            break;
        case 'L':
            x[0] = x[0] - UNIT_SIZE;
            break;
        case 'R':
            x[0] = x[0] + UNIT_SIZE;
            break;
    }
}

public void checkApple() {
    if ((x[0] == appleX) && (y[0] == appleY)) {
        bodyParts++;
        score++;
        newApple();
    }
}

public void checkCollisions() {
    // Checa colisão com a parede
    if ((x[0] >= SCREEN_WIDTH) || (x[0] < 0) || (y[0] >= SCREEN_HEIGHT) || (y[0] < 0)) {
        running = false;
    }

    // Checa colisão com o corpo
    for (int i = bodyParts; i > 0; i--) {
        if ((x[0] == x[i]) && (y[0] == y[i])) {
            running = false;
        }
    }
```

```
        if (!running) {
            timer.stop();
            gameOver();
        }
    }

    public void gameOver() {
        removeAll(); // Remove todos os componentes do painel
        isStartScreen = false; // Indica que o jogo não está na tela de início

        restartButton = new JButton("Reiniciar");
        restartButton.addActionListener(e -> startGame());
        add(restartButton);

        repaint();
    }

    @Override
    public void keyPressed(KeyEvent e) {
        switch (e.getKeyCode()) {
            case KeyEvent.VK_LEFT:
            case KeyEvent.VK_A:
                if (direction != 'R') {
                    direction = 'L';
                }
                break;
                case KeyEvent.VK_RIGHT:
            case KeyEvent.VK_D:
                if (direction != 'L') {
                    direction = 'R';
                }
                break;
            case KeyEvent.VK_UP:
            case KeyEvent.VK_W:
                if (direction != 'D') {
                    direction = 'U';
                }
                break;
            case KeyEvent.VK_DOWN:
            case KeyEvent.VK_S:
                if (direction != 'U') {
                    direction = 'D';
                }
                break;
        }
    }

    @Override
    public void keyTyped(KeyEvent e) {}

    @Override
    public void keyReleased(KeyEvent e) {}


    public static void main(String[] args) {
        JFrame frame = new JFrame("Snake Game");
        SnakeGame snakeGame = new SnakeGame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setExtendedState(JFrame.MAXIMIZED_BOTH); // Define a janela para tela cheia
        frame.setResizable(false);
        frame.add(snakeGame);
        frame.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (running) {
            move();
            checkApple();
            checkCollisions();
        }
        repaint();
    }
}
```

O código implementa o clássico jogo da cobrinha em Java com interface gráfica utilizando Swing. A classe `SnakeGame` é uma extensão de `JPanel` e implementa `ActionListener` e `KeyListener, que` controla o jogo, incluindo a lógica da cobra, a maçã, a detecção de colisões e a interface gráfica. As dimensões da tela são obtidas a partir do tamanho da tela do dispositivo, tendo a cobra representada por uma série de

coordenadas x e y. A maça é gerada aleatoriamente e vai aumentando o tamanho da cobra uma vez que consumida. A cobra é direcionada através das teclas W, A, S, D ou pelas teclas de seta. O jogo é iniciado e reiniciado por meio de botões na tela e o método `actionPerformed` é chamado a cada intervalo de tempo definido pelo temporizador e controla o movimento da cobra, a verificação de colisões e a atualização da tela.