import javax.swing.\*;

public class App {

public static void main(String[] args) throws Exception {

int boardWidth = 600;

int boardHeight = boardWidth;

JFrame frame = new JFrame("Snake Game");

frame.setVisible(true);

frame.setSize(boardWidth, boardHeight);

frame.setLocationRelativeTo(null);

frame.setResizable(false);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

SnakeGamee snakeGamee = new SnakeGamee(boardWidth, boardHeight);

frame.add(snakeGamee);

frame.pack();

snakeGamee.requestFocus();

}

}

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Random;

import javax.swing.\*;

public class SnakeGamee extends JPanel implements ActionListener, KeyListener {

private class Tile {

int x;

int y;

Tile(int x, int y) {

this.x = x;

this.y = y;

}

}

int boardWidth;

int boardHeight;

int tileSize = 25;

Tile snakeHead;

ArrayList<Tile> snakeBody;

Tile food;

Random random;

// Game logic

Timer gameLoop;

int velocityX;

int velocityY;

boolean gameOver = false;

// Score and high score

int score = 0;

int highScore = 0;

// Username

String username;

// Buttons

private JButton playAgainButton;

private JButton quitButton;

// Flag to prevent multiple username prompts

private boolean usernamePromptShown = false;

SnakeGamee(int boardWidth, int boardHeight) {

this.boardWidth = boardWidth;

this.boardHeight = boardHeight;

setPreferredSize(new Dimension(this.boardWidth, this.boardHeight));

setBackground(Color.black);

addKeyListener(this);

setFocusable(true);

// Ask for username before starting the game (only once)

if (!usernamePromptShown) {

username = JOptionPane.showInputDialog(null, "Enter your username:", "Player Name", JOptionPane.PLAIN\_MESSAGE);

usernamePromptShown = true;

}

// Initialize the game

snakeHead = new Tile(5, 5);

snakeBody = new ArrayList<Tile>();

food = new Tile(10, 10);

random = new Random();

placeFood();

velocityX = 0;

velocityY = 0;

gameLoop = new Timer(100, this);

gameLoop.start();

// Play Again Button

playAgainButton = new JButton("Restart");

playAgainButton.setBounds(boardWidth / 2 - 110, boardHeight / 2 + 50, 100, 40); // Restart on left

playAgainButton.setBackground(Color.green);

playAgainButton.setForeground(Color.white);

playAgainButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

restartGame();

}

});

// Quit Button

quitButton = new JButton("Quit Game");

quitButton.setBounds(boardWidth / 2 + 10, boardHeight / 2 + 50, 100, 40); // Quit on right

quitButton.setBackground(Color.red);

quitButton.setForeground(Color.white);

quitButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.exit(0); // This ensures the entire program quits

}

});

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

draw(g);

}

public void draw(Graphics g) {

// Check for game over condition and display the "Game Over" message

if (gameOver) {

g.setColor(Color.black); // Set the background to black

g.fillRect(0, 0, boardWidth, boardHeight); // Fill the entire frame with black

// Show Highest Score (smaller and above the current score)

g.setFont(new Font("Arial", Font.PLAIN, 18)); // Smaller font for highest score

g.setColor(Color.white);

String highestScoreText = "Highest Score: " + highScore;

FontMetrics fmHighScore = g.getFontMetrics();

int xHighScore = (boardWidth - fmHighScore.stringWidth(highestScoreText)) / 2;

int yHighScore = boardHeight / 2 - 100; // Position above the score

g.drawString(highestScoreText, xHighScore, yHighScore);

// Show current score (larger and below highest score)

g.setFont(new Font("Arial", Font.PLAIN, 20)); // Larger font for current score

String currentScoreText = "Score: " + score;

FontMetrics fmScore = g.getFontMetrics();

int xScore = (boardWidth - fmScore.stringWidth(currentScoreText)) / 2;

int yScore = boardHeight / 2 - 60; // Position below the highest score

g.drawString(currentScoreText, xScore, yScore);

// Draw "Game Over" text in a large font in the center of the screen

g.setColor(Color.white); // Set the color of the text to white

g.setFont(new Font("Arial", Font.BOLD, 50)); // Set a large font

String gameOverMessage = "Game Over";

FontMetrics fm = g.getFontMetrics();

int x = (boardWidth - fm.stringWidth(gameOverMessage)) / 2; // Center horizontally

int y = (boardHeight - fm.getHeight()) / 2 + fm.getAscent(); // Center vertically

g.drawString(gameOverMessage, x, y); // Draw the "Game Over" message

// Add the "Play Again" and "Quit Game" buttons

add(playAgainButton);

add(quitButton);

} else {

// If the game is not over, continue drawing the snake and food

// Food

g.setColor(Color.red);

g.fillRect(food.x \* tileSize, food.y \* tileSize, tileSize, tileSize);

// Snake Head

g.setColor(Color.green);

g.fillRect(snakeHead.x \* tileSize, snakeHead.y \* tileSize, tileSize, tileSize);

// Snake Body

for (int i = 0; i < snakeBody.size(); i++) {

Tile snakePart = snakeBody.get(i);

g.fillRect(snakePart.x \* tileSize, snakePart.y \* tileSize, tileSize, tileSize);

}

// Display the username at the top

g.setFont(new Font("Arial", Font.PLAIN, 16));

g.setColor(Color.white);

g.drawString("Player: " + username, 10, 20);

// Display Highest Score below the username (on the left)

g.setFont(new Font("Arial", Font.PLAIN, 14)); // Smaller font for highest score

String highestScoreText = "Highest Score: " + highScore;

g.setColor(Color.white);

g.drawString(highestScoreText, 10, 40); // Position below the username

}

}

public void placeFood() {

food.x = random.nextInt(boardWidth / tileSize); // 600 / 25 = 24

food.y = random.nextInt(boardHeight / tileSize);

}

public boolean collision(Tile tile1, Tile tile2) {

return tile1.x == tile2.x && tile1.y == tile2.y;

}

public void move() {

// Eat food

if (collision(snakeHead, food)) {

snakeBody.add(new Tile(food.x, food.y));

placeFood();

score++; // Increase score on eating food

}

// Snake Body

for (int i = snakeBody.size() - 1; i >= 0; i--) {

Tile snakePart = snakeBody.get(i);

if (i == 0) {

snakePart.x = snakeHead.x;

snakePart.y = snakeHead.y;

} else {

Tile prevSnakePart = snakeBody.get(i - 1);

snakePart.x = prevSnakePart.x;

snakePart.y = prevSnakePart.y;

}

}

// Snake Head

snakeHead.x += velocityX;

snakeHead.y += velocityY;

// Game Over Conditions

for (int i = 0; i < snakeBody.size(); i++) {

Tile snakePart = snakeBody.get(i);

// If Collide with Snake Head

if (collision(snakeHead, snakePart)) {

gameOver = true;

}

}

if (snakeHead.x \* tileSize < 0 || snakeHead.x \* tileSize > boardWidth ||

snakeHead.y \* tileSize < 0 || snakeHead.y \* tileSize > boardHeight) {

gameOver = true;

}

}

@Override

public void actionPerformed(ActionEvent e) {

move();

repaint();

if (gameOver) {

// Update high score if necessary

if (score > highScore) {

highScore = score;

}

gameLoop.stop();

}

}

@Override

public void keyPressed(KeyEvent e) {

if (e.getKeyCode() == KeyEvent.VK\_UP && velocityY != 1) {

velocityX = 0;

velocityY = -1;

} else if (e.getKeyCode() == KeyEvent.VK\_DOWN && velocityY != -1) {

velocityX = 0;

velocityY = 1;

} else if (e.getKeyCode() == KeyEvent.VK\_LEFT && velocityX != 1) {

velocityX = -1;

velocityY = 0;

} else if (e.getKeyCode() == KeyEvent.VK\_RIGHT && velocityX != -1) {

velocityX = 1;

velocityY = 0;

}

}

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyReleased(KeyEvent e) {

}

// Method to reset the game when the "Play Again" button is clicked

private void restartGame() {

// Reset game state

snakeHead = new Tile(5, 5);

snakeBody.clear();

food = new Tile(10, 10);

random = new Random();

placeFood();

velocityX = 0;

velocityY = 0;

score = 0;

gameOver = false;

// Start the game loop again

gameLoop.start();

// Re-add the key listener to ensure it works after restart

addKeyListener(this);

setFocusable(true);

// Remove buttons after restart

remove(playAgainButton);

remove(quitButton);

// Request focus to listen for key events

requestFocus();

// Repaint the screen

repaint();

}

}