TASK- 1

Task List Application

SOURCE CODE:

//START OF CODE:

mport javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.util.Random;

public class SnakeGame extends JFrame implements ActionListener, KeyListener {

private static final int GRID\_SIZE = 20;

private static final int CELL\_SIZE = 20;

private static final int GAME\_SPEED = 150; // Milliseconds per frame

private enum Direction {

UP, DOWN, LEFT, RIGHT

}

private Direction currentDirection = Direction.RIGHT;

private Direction nextDirection = Direction.RIGHT;

private boolean running = false;

private Timer timer;

private int[] snakeX, snakeY;

private int snakeLength;

private int foodX, foodY;

public SnakeGame() {

setTitle("Snake Game");

setSize(GRID\_SIZE \* CELL\_SIZE, GRID\_SIZE \* CELL\_SIZE);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

snakeX = new int[GRID\_SIZE \* GRID\_SIZE];

snakeY = new int[GRID\_SIZE \* GRID\_SIZE];

addKeyListener(this);

setFocusable(true);

timer = new Timer(GAME\_SPEED, this);

initializeGame();

}

private void initializeGame() {

running = true;

snakeLength = 3;

snakeX[0] = 3;

snakeY[0] = 1;

snakeX[1] = 2;

snakeY[1] = 1;

snakeX[2] = 1;

snakeY[2] = 1;

spawnFood();

timer.start();

}

private void spawnFood() {

Random random = new Random();

foodX = random.nextInt(GRID\_SIZE);

foodY = random.nextInt(GRID\_SIZE);

// Ensure food doesn't spawn on the snake

while (isFoodOnSnake()) {

foodX = random.nextInt(GRID\_SIZE);

foodY = random.nextInt(GRID\_SIZE);

}

}

private boolean isFoodOnSnake() {

for (int i = 0; i < snakeLength; i++) {

if (snakeX[i] == foodX && snakeY[i] == foodY) {

return true;

}

}

return false;

}

private void move() {

// Move the body of the snake

for (int i = snakeLength - 1; i > 0; i--) {

snakeX[i] = snakeX[i - 1];

snakeY[i] = snakeY[i - 1];

}

// Move the head of the snake based on the current direction

switch (currentDirection) {

case UP:

snakeY[0]--;

break;

case DOWN:

snakeY[0]++;

break;

case LEFT:

snakeX[0]--;

break;

case RIGHT:

snakeX[0]++;

break;

}

// Check for collisions

checkCollisions();

}

private void checkCollisions() {

// Check if the snake hits the walls

if (snakeX[0] < 0 || snakeX[0] >= GRID\_SIZE || snakeY[0] < 0 || snakeY[0] >= GRID\_SIZE) {

gameOver();

return;

}

// Check if the snake eats the food

if (snakeX[0] == foodX && snakeY[0] == foodY) {

snakeLength++;

spawnFood();

}

// Check if the snake collides with itself

for (int i = 1; i < snakeLength; i++) {

if (snakeX[0] == snakeX[i] && snakeY[0] == snakeY[i]) {

gameOver();

return;

}

}

}

private void gameOver() {

running = false;

timer.stop();

JOptionPane.showMessageDialog(this, "Game Over! Your score: " + (snakeLength - 3),

"Game Over", JOptionPane.INFORMATION\_MESSAGE);

initializeGame();

}

private void draw(Graphics g) {

// Draw the snake

g.setColor(Color.GREEN);

for (int i = 0; i < snakeLength; i++) {

g.fillRect(snakeX[i] \* CELL\_SIZE, snakeY[i] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

}

// Draw the food

g.setColor(Color.RED);

g.fillRect(foodX \* CELL\_SIZE, foodY \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

}

@Override

public void actionPerformed(ActionEvent e) {

if (running) {

move();

repaint();

}

}

@Override

public void keyPressed(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_UP:

if (currentDirection != Direction.DOWN) {

nextDirection = Direction.UP;

}

break;

case KeyEvent.VK\_DOWN:

if (currentDirection != Direction.UP) {

nextDirection = Direction.DOWN;

}

break;

case KeyEvent.VK\_LEFT:

if (currentDirection != Direction.RIGHT) {

nextDirection = Direction.LEFT;

}

break;

case KeyEvent.VK\_RIGHT:

if (currentDirection != Direction.LEFT) {

nextDirection = Direction.RIGHT;

}

break;

}

}

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void keyReleased(KeyEvent e) {

}

@Override

public void paint(Graphics g) {

super.paint(g);

draw(g);

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

SnakeGame snakeGame = new SnakeGame();

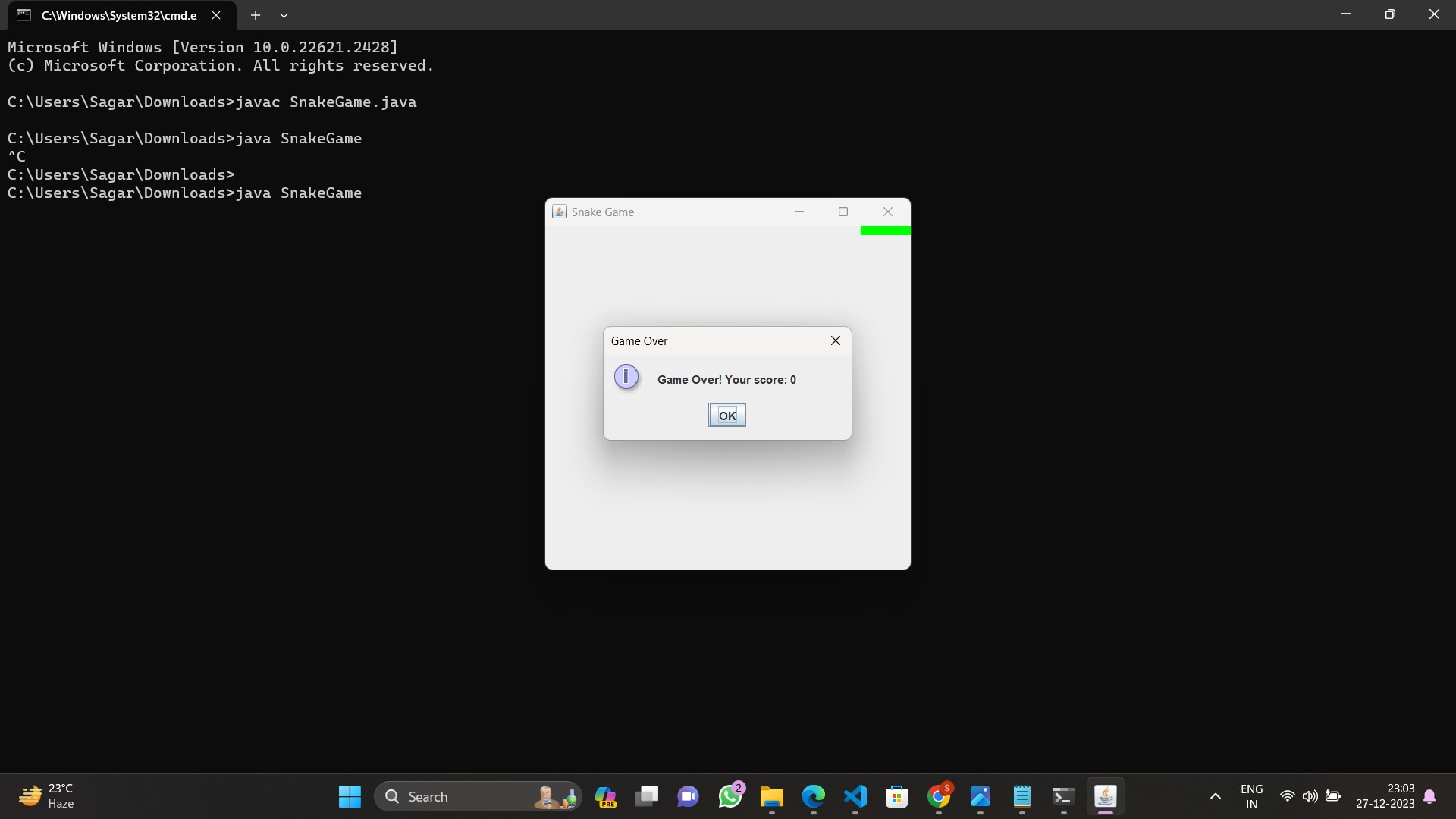
snakeGame.setVisible(true);

});

}

}

**output:**



**EXPLANATION:**

**1. Class Hierarchy:**

**The code uses a class hierarchy to structure the game. There is a base class SnakeGame that contains the core functionality of the game, and an enhanced class EnhancedSnakeGame that inherits from SnakeGame to add extra features.**

**2. Score and Level:**

**The enhanced game introduces a scoring system and levels. The score variable keeps track of the player's score, and the level variable tracks the current level. The updateScore() and updateLevel() methods are responsible for updating the game window title to display the current score and level.**

**3. Method Overrides:**

**The initializeGame(), spawnFood(), and gameOver() methods are overridden in the EnhancedSnakeGame class to provide enhanced functionality. For example, the spawnFood() method increments the score when the snake eats food and updates the level based on the score.**

**4. JOptionPane for Game Over Message:**

**When the game is over, a JOptionPane is used to display a message dialog with the player's final score. This provides a user-friendly way to communicate game-over information.**

**5. Ongoing Enhancements:**

**The code is designed to be extensible. You can further enhance the game by adding more features, such as obstacle handling, customization options, sound effects, pause/resume functionality, smooth animations, and high score tracking.**

**6. Object-Oriented Principles:**

**The code adheres to object-oriented principles by using classes and inheritance. This makes the code modular and easier to maintain. Additionally, the code utilizes encapsulation, as the internal details of the game state are managed within the class methods.**

**7. Swing and GUI:**

**The game utilizes the Java Swing library for creating a graphical user interface (GUI). It includes components like JFrame for the main window, Graphics for drawing elements, and JOptionPane for displaying messages.**

**8. Game Loop:**

**The game loop is handled by the Timer class, which triggers the actionPerformed() method at regular intervals. This method is responsible for updating the game state, handling user input, and repainting the screen.**

**9. Event Handling:**

**User input is captured using the KeyListener interface. Arrow key presses are detected in the keyPressed() method, and the snake's direction is updated accordingly.**

**10. Customization Points:**

**The code provides customization points where you can extend or modify behavior. For example, you can add more enhancements, change scoring rules, or customize the appearance of the game.**

**Overall, this code provides a foundation for a Snake game with room for expansion and customization. You can continue to build upon it to create a more sophisticated and enjoyable gaming experience.**

**---------------------------------------------END--------------------------------------------------**