# *JAVA CASE STUDY-Snake Game*

***VU22CSEN0100120***

***Sai Mrudula.D***

## Problem Statement :

Create an engaging and nostalgic gaming experience by developing a modern rendition of the classic Snake game in Java. The Snake game is a beloved childhood favourite that has captivated generations with its simple yet addictive gameplay. By harnessing the power of programming, this project aims to breathe new life into this timeless classic, providing players with an immersive and entertaining gaming experience.

## Introduction:

The Snake game is a classic arcade game where the player controls a snake that moves around the game board, eating food to grow in length. This implementation of the Snake game is built in Java using the Swing library for the graphical user interface. Players navigate the snake using arrow keys, attempting to eat as much food as possible without colliding with the walls or the snake itself.

## Design and Implementation:

### Design:

- **Game Mechanics:** Implement the core mechanics of the Snake game, including snake movement, apple generation, and collision detection.
- **User Interface:** Design an intuitive and visually appealing user interface using Java Swing components to provide players with an immersive gaming experience.
- **Scoring System:** Develop a scoring system to track the player's progress and provide feedback on their performance.
- **Game Over Screen:** Display a game over screen with the final score and an option to restart the game, allowing players to continue playing and improving their skills.

### Implementation:

- **Game Panel Class:** Create a Game Panel class responsible for managing the game state, user input, and rendering the game graphics.
- **Timer Integration:** Integrate a timer to regulate game updates and ensure smooth gameplay.
- **Event Handling:** Implement event handlers to capture user input and control the snake's movement based on keyboard commands.
- **Random Apple Generation:** Develop a mechanism to randomly generate apples on the game board, providing players with opportunities to increase their score.
- **Collision Detection:** Implement collision detection algorithms to detect collisions between the snake and walls or itself, triggering game over scenarios when necessary.
- **Score Tracking:** Implement a scoring system to keep track of apples eaten and display the player's score on the game screen.
- **Game Over Handling:** Display a game over screen when the game ends, allowing players to view their final score and choose whether to restart the game.

### Fields:

- **SCREEN_WIDTH** and **SCREEN_HEIGHT:** Define the dimensions of the game window.
- **UNIT_SIZE:** Specify the size of each unit on the game board.

- **GAME_UNITS:** Calculate the total number of units on the game board.
- **DELAY:** Determine the delay between each game update.
- **x[]** and **y[]:** Arrays to store the coordinates of the snake's segments.
- **bodyParts:** Track the number of segments in the snake's body.
- **applesEaten:** Count the number of apples eaten by the snake.
- **appleX** and **appleY:** Store the coordinates of the apple.
- **direction:** Store the current direction of the snake.
- **running:** Boolean flag to indicate whether the game is running.
- **timer:** Timer object to control game updates.
- **random:** Random object to generate random positions for the apple.

## ***Methods:***

- **startGame():** Initializes the game state and starts the game loop.
- **paintComponent(Graphics g):** Draws the game elements on the panel.
- **draw(Graphics g):** Draws the snake, apple, and score on the panel.
- **newApple():** Generates a new random position for the apple.
- **move():** Moves the snake in the current direction.
- **checkApple():** Checks if the snake has eaten the apple and updates the score.
- **checkCollisions():** Checks for collisions between the snake and the walls or itself.
- **gameOver(Graphics g):** Displays the game over screen with the final score.
- **actionPerformed(ActionEvent e):** Handles game updates triggered by the timer.
- **MyKeyAdapter:** Inner class to handle keyboard input for changing the snake's direction.

## ***Code:***

```
//**********************************
```

***Class Name : SnakeGame***

```
public class SnakeGame {

        public static void main(String[] args) {

                new GameFrame();
        }
}
//*****************************************
```

***Class Name : SnakeFrame***

```
import javax.swing.JFrame;

public class GameFrame extends JFrame{

        GameFrame(){

                this.add(new GamePanel());
                this.setTitle("Snake");
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setResizable(false);
                this.pack();
                this.setVisible(true);
                this.setLocationRelativeTo(null);

        }
}
//****************************************
```

***Class Name : SnakePanel***

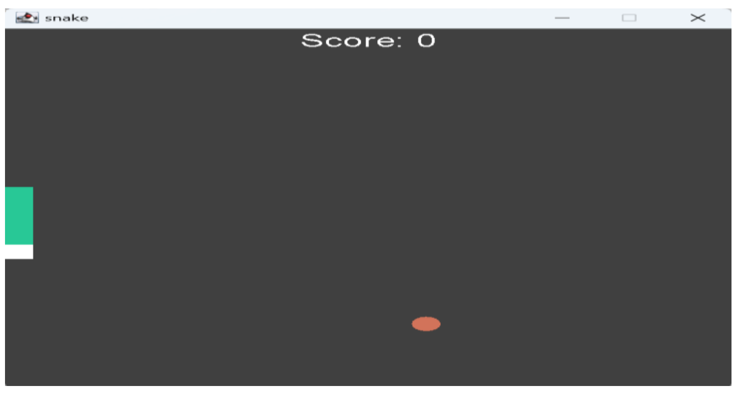
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;

public class GamePanel extends JPanel implements ActionListener{

    static final int SCREEN_WIDTH = 1300;
    static final int SCREEN_HEIGHT = 750;
    static final int UNIT_SIZE = 50;
    static final int GAME_UNITS = (SCREEN_WIDTH*SCREEN_HEIGHT)/(UNIT_SIZE*UNIT_SIZE);
    static final int DELAY = 175;
    final int x[] = new int[GAME_UNITS];
    final int y[] = new int[GAME_UNITS];
    int bodyParts = 6;
    int applesEaten;
    int appleX;
    int appleY;
    char direction = 'R';
    boolean running = false;
    Timer timer;
    Random random;

    GamePanel(){
        random = new Random();
        this.setPreferredSize(new Dimension(SCREEN_WIDTH,SCREEN_HEIGHT));
        this.setBackground(Color.black);
        this.setFocusable(true);
        this.addKeyListener(new MyKeyAdapter());
        startGame();
    }
    public void startGame() {
        newApple();
        running = true;
        timer = new Timer(DELAY,this);
        timer.start();
    }
```

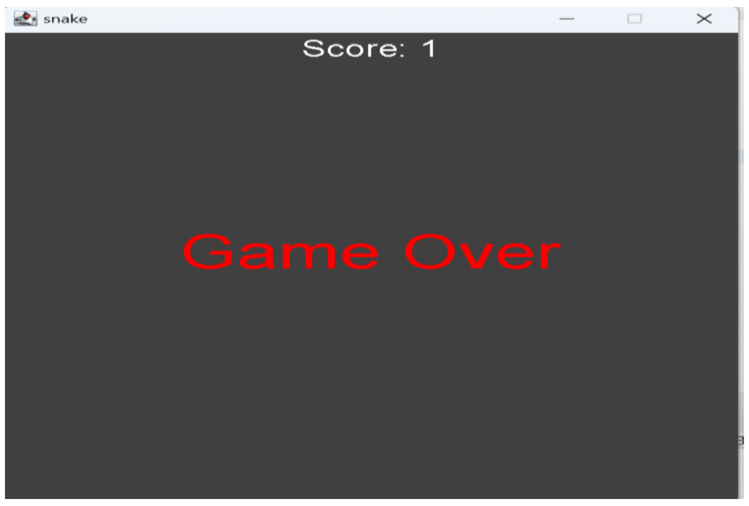
```
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		draw(g);
	}
	public void draw(Graphics g) {

		if(running) {
			/*
			for(int i=0;i<SCREEN_HEIGHT/UNIT_SIZE;i++) {
				g.drawLine(i*UNIT_SIZE, 0, i*UNIT_SIZE, SCREEN_HEIGHT);
				g.drawLine(0, i*UNIT_SIZE, SCREEN_WIDTH, i*UNIT_SIZE);
			}
			*/
			g.setColor(Color.red);
			g.fillOval(appleX, appleY, UNIT_SIZE, UNIT_SIZE);

			for(int i = 0; i< bodyParts;i++) {
				if(i == 0) {
					g.setColor(Color.green);
					g.fillRect(x[i], y[i], UNIT_SIZE, UNIT_SIZE);
				}
				else {
					g.setColor(new Color(45,180,0));
					//g.setColor(new Color(random.nextInt(255),random.nextInt(255),random.nextInt(255)));
					g.fillRect(x[i], y[i], UNIT_SIZE, UNIT_SIZE);
				}
			}
			g.setColor(Color.red);
			g.setFont( new Font("Ink Free",Font.BOLD, 40));
			FontMetrics metrics = getFontMetrics(g.getFont());
			g.drawString("Score: "+applesEaten, (SCREEN_WIDTH - metrics.stringWidth("Score: "+applesEaten))/2,
g.getFont().getSize());
		}
		else {
			gameOver(g);
		}

	}
	public void newApple(){
		appleX = random.nextInt((int)(SCREEN_WIDTH/UNIT_SIZE))*UNIT_SIZE;
		appleY = random.nextInt((int)(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++;
                applesEaten++;
                newApple();
        }
}
public void checkCollisions() {
        //checks if head collides with body
        for(int i = bodyParts;i>0;i--) {
                if((x[0] == x[i])&& (y[0] == y[i])) {
                        running = false;
                }
        }
        //check if head touches left border
        if(x[0] < 0) {
                running = false;
        }
        //check if head touches right border
        if(x[0] > SCREEN_WIDTH) {
                running = false;
        }
        //check if head touches top border
        if(y[0] < 0) {
                running = false;
        }
        //check if head touches bottom border
        if(y[0] > SCREEN_HEIGHT) {
                running = false;
        }

        if(!running) {
                timer.stop();
        }
}
public void gameOver(Graphics g) {
        //Score
        g.setColor(Color.red);
        g.setFont( new Font("Ink Free",Font.BOLD, 40));
        FontMetrics metrics1 = getFontMetrics(g.getFont());
        g.drawString("Score: "+applesEaten, (SCREEN_WIDTH - metrics1.stringWidth("Score: "+applesEaten))/2,
g.getFont().getSize());
        //Game Over text
        g.setColor(Color.red);
        g.setFont( new Font("Ink Free",Font.BOLD, 75));
        FontMetrics metrics2 = getFontMetrics(g.getFont());
        g.drawString("Game Over", (SCREEN_WIDTH - metrics2.stringWidth("Game Over"))/2, SCREEN_HEIGHT/2);
}
@Override
public void actionPerformed(ActionEvent e) {

        if(running) {
                move();
                checkApple();
                checkCollisions();
        }
        repaint();
}

public class MyKeyAdapter extends KeyAdapter{
        @Override
        public void keyPressed(KeyEvent e) {
                switch(e.getKeyCode()) {
                case KeyEvent.VK_LEFT:
                        if(direction != 'R') {
                            direction = 'L';
                        }
                        break;
                case KeyEvent.VK_RIGHT:
                        if(direction != 'L') {
                            direction = 'R';
                        }
                        break;
                case KeyEvent.VK_UP:
                        if(direction != 'D') {
                            direction = 'U';
                        }
                        break;
                case KeyEvent.VK_DOWN:
                        if(direction != 'U') {
```

```
                                        direction = 'D';
                                }
                                break;
                        }
                }
        }
}
```

## Output :





## Conclusion:

Through the development of this Java-based Snake game, we aim to provide players with a captivating and enjoyable gaming experience that pays homage to a beloved classic. By combining nostalgic gameplay elements with modern programming techniques, we hope to create a game that appeals to players of all ages and backgrounds, fostering a sense of excitement and nostalgia for generations to come.