National Institute of Technology, Kurukshetra

Department of Computer Applications

Semester long project

for

(MCA-104) Object Oriented Programming using Java

Titled

Snake Game

Submitted By

Ankush Jain (523110053)

Akash Kumar (523410019)

Shreyash Badole (523110056)

Under the guidance of

Dr Kapil Gupta

CERTIFICATE

This is to certify that the semester long project report titled “Snake Game” submitted by ANKUSH JAIN, AKASH KUMAR AND SHREYASH BADOLE has been carried out under the guidance of Dr. Kapil Gupta, Master of computer application, National Institute Of Technology, Kurukshetra. The project report is approved for submission requirement for Semester long Project in 2nd semester in Master of Computer Application, National Institute Of Technology, Kurukshetra during the academic session JAN-MAY, 2024.

GUIDED BY
Dr. Kapil Gupta

Acknowledgement

I would like to express my heartfelt gratitude to my teacher, Dr. Kapil Gupta, for providing me with the opportunity to work on this wonderful project of creating a snake game. His guidance has been invaluable throughout the development of this project, and I am truly grateful for his support.

Date:
19/04/2024

Team Members:
Ankush Jain
Akash Kumar
Shreyash Badole

Contents

Introduction

The Snake Game is a classic arcade game where the player controls a snake that moves around a board, eating food pellets to grow larger while avoiding collisions with its own tail and body.

This project aims to implement a simple version of the Snake Game using Java and the Swing framework for GUI development.

This project report delves into the intricacies of our implementation, detailing the features, design principles, and technical aspects that contribute to the game's functionality and appeal. From object-oriented analysis to code snippets, we provide a comprehensive overview of our development process.

This game requires the player to use the function keys for basic movements such as up, right, down, and left. While most games do not allow the player to change directions from right to left or up to down in one step, this game allows it. To prevent this, I have implemented a temporary direction that reads the current direction at the beginning of the video. This ensures that the player cannot change direction in one step.

HARDWARE & SOFTWARE REQUIREMENTS

HARDWARE REQUIRED

Number	Description
1.	Intel i3 10^{th} gen or above
2.	4096 MB RAM

SOFTWARE REQUIRED

Number	Description
1.	Windows 10,11
2.	Java version "21.0.2" 2024-01-16 LTS
3.	Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
5.	VS Code, Eclipse IDE

Technologies

The Snake Game project leverages several technologies to bring the classic arcade experience to life in a modern context:

1. Java: The core programming language used for development, offering platform independence, robustness, and object-oriented capabilities.

2. Swing Framework: A GUI (Graphical User Interface) toolkit for Java, used to create the game's graphical interface, including windows, buttons, and interactive elements.

3. ImageIcon Class: Part of the Java Swing toolkit, ImageIcon is used to load and display images in the game, such as the snake, food, and game elements.

4. Timer Class: Provided by the Java Swing framework, Timer is utilized to control the game's animation and update cycle, ensuring smooth gameplay.

5. Random Class: A utility class in Java used to generate random numbers, employed in the project to randomize the positions of food items (enemies) on the game board.

6. IDE (Integrated Development Environment): While not explicitly mentioned, an IDE such as Eclipse and VS code is likely used for coding, debugging, and managing the project files.

7. Graphics Programming: The project involves graphics programming techniques to draw and render game elements dynamically on the screen, enhancing the visual appeal and interactivity of the game.

These technologies collectively enable the implementation of key features such as snake movement, food generation, scoring, collision detection, and user interaction, culminating in an engaging and enjoyable gaming experience.

Features

- Snake Movement: The snake can be controlled using arrow keys to move in different directions.
- Food Generation: Randomly positioned food items appear on the board for the snake to eat.
- Scoring: The player earns points for each food item consumed.
- Collision Detection: The game detects collisions between the snake's head and its body, resulting in a game over.
- Restart Option: The player can restart the game by pressing the spacebar after the game is over.

Implementation

Classes

- Gameplay: This class extends `JPanel` and implements `KeyListener` and `ActionListener`. It represents the main gameplay area where the snake moves and interacts with food and obstacles.
- Main: This class contains the `main` method and is responsible for initializing the game window and starting the gameplay.

Variables

- Snake Length: Arrays `xSnakeLength` and `ySnakeLength` store the coordinates of each segment of the snake's body.
- Direction: Booleans `rightDirection`, `leftDirection`, `upDirection`, and `downDirection` determine the direction in which the snake moves.
- Enemy Positions: Arrays `xEnemyPos` and `yEnemyPos` store the positions where food items (enemies) can appear.
- Timer: A `Timer` object controls the game's animation and updates.
- Score: An integer variable `score` keeps track of the player's score.
- Images: `ImageIcon` objects represent graphical elements such as the snake, food, and game title.

Methods

- Paint (Graphics g): Draws the game elements including the snake, food, score, and game over message.
- actionPerformed(ActionEvent e): Handles game updates and movements triggered by the timer.
- keyPressed(KeyEvent e): Responds to keyboard input for controlling the snake's direction and restarting the game.
- keyReleased(KeyEvent e), keyTyped(KeyEvent e): Unused methods required by the `KeyListener` interface.

Dependencies

The project relies on external image files (‘snaketitle.jpg’, ‘rightmouth.png’, ‘leftmouth.png’, ‘upmouth.png’, ‘downmouth.png’, ‘snakeimage.png’, ‘enemy.png’) for visual assets.

Object-Oriented Analysis:

Classes:

i. **Gameplay**: Represents the gameplay area where the snake moves and interacts with food and obstacles.

- Responsibilities:
    - Handling user input for controlling the snake's movement. o Updating the game state based on user input and timer events. o Drawing the game elements on the screen.

- OOP Concepts Applied:
    - Encapsulation: The `Gameplay` class encapsulates the game's logic, including movement, collision detection, and rendering. o Inheritance: Extends `JPanel` to utilize Swing's GUI capabilities.
    - Composition: Uses `Timer` for managing game updates and drawing.

ii. **Main**: Contains the `main` method and initializes the game window and gameplay.

- Responsibilities:
    - Start the game by creating an instance of the `Gameplay` class.

- OOP Concepts Applied:
    - Encapsulation: The `Main` class encapsulates the entry point and initialization logic of the game.

Object-Oriented Design:

i Encapsulation:

- All variables and methods in the `Gameplay` class are encapsulated within the class, providing data hiding and preventing direct access from outside.
- Methods like ‘paint(Graphics g)’, `actionPerformed(ActionEvent e)`, and `keyPressed(KeyEvent e)` encapsulate specific functionalities related to drawing, game updates, and input handling, respectively.

ii Inheritance:

- The `Gameplay` class inherits from `JPanel`, leveraging its functionality for creating a graphical user interface.

iii Composition:

- The `Gameplay` class uses composition to include a `Timer` object (`timer`) for managing game updates and triggering repaint events.

iv Abstraction:

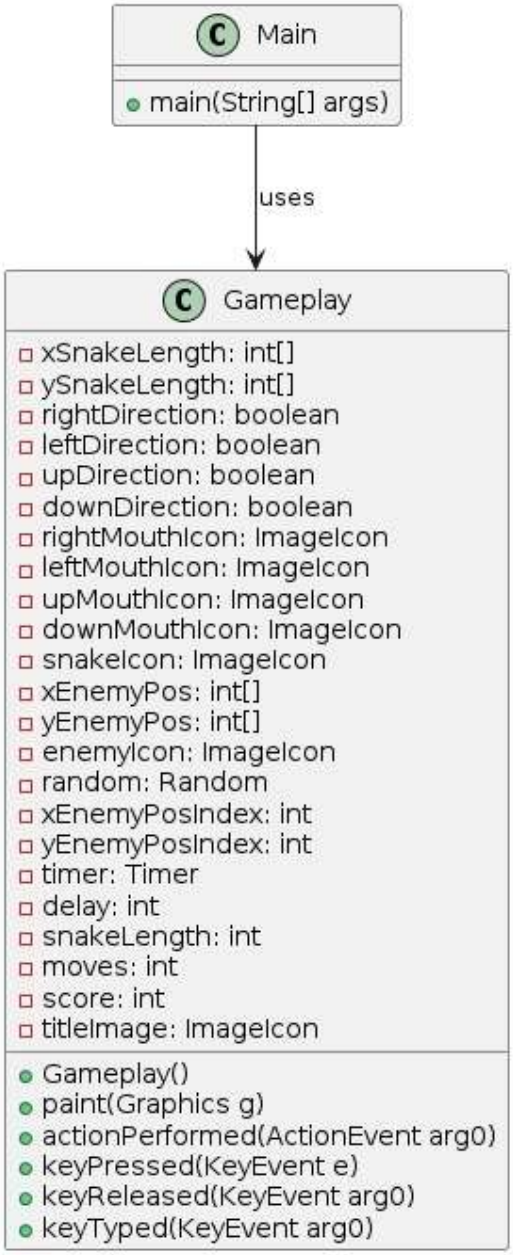
- The `Gameplay` class abstracts away the details of drawing game elements and handling user input. It provides a high-level interface for managing the game's state and behaviour.

v Polymorphism:

- The `paint(Graphics g)` method in the `Gameplay` class demonstrates polymorphism by dynamically rendering di erent game elements based on their current state and position.

Applying object-oriented principles improves modularity, maintainability, and extensibility. It separates concerns, encapsulates behaviour, and simplifies understanding. Using object-oriented principles makes things easier to manage, maintain and develop. This approach separates di erent things, keeps their behaviour in check, and simplifies understanding.

Class Diagram: -



Main.java Code: -

```
import java.awt.Color;

import javax.swing.JFrame;

public class Main {

	public static void main(String[] args) {

		JFrame f = new JFrame();
		f.setTitle("Snake Game");
		f.setBounds(10,10,905,700);
		f.setResizable(false);
		f.setVisible(true);
		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		f.setBackground(Color.DARK_GRAY);

		Gameplay game = new Gameplay();
		f.add(game);


	}

}
```

Gameplay.java Code: -

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.Random;

import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.Timer;

public class Gameplay extends JPanel implements KeyListener, ActionListener{

	private int[] xSnakeLength = new int[750];
	private int[] ySnakeLength = new int[750];

	private boolean rightDirection = false;
	private boolean leftDirection = false;
	private boolean upDirection = false;
	private boolean downDirection = false;

	private ImageIcon rightMouthIcon;
	private ImageIcon leftMouthIcon;
	private ImageIcon upMouthIcon;
	private ImageIcon downMouthIcon;
	private ImageIcon snakeIcon;

	private int[] xEnemyPos =
{25,50,75,100,125,150,175,200,225,250,275,300,325,350,375,400,425,450,475,
500,525,550,575,
			600,625,650,675,700,725,750,775,800,825,850};
```

```
        private int[] yEnemyPos =
{75,100,125,150,175,200,225,250,275,300,325,350,375,400,425,450,475,500,5
25,550,575,
                        600,625};

        private ImageIcon enemyIcon;

        private Random random=new Random();

        private int xEnemyPosIndex = random.nextInt(34);
        private int yEnemyPosIndex = random.nextInt(23);


        private Timer timer;
        private int delay=90;

        private int snakeLength=3;
        private int moves=0;
        private int score=0;

        private ImageIcon titleImage;

        public Gameplay() {
                addKeyListener(this);
                setFocusable(true);
                setFocusTraversalKeysEnabled(true);
                timer=new Timer(delay,this);
                timer.start();
        }
        public void paint(Graphics g) {

                if(moves==0)
                {
                        xSnakeLength[0]=100;
                        xSnakeLength[1]=75;
                        xSnakeLength[2]=50;

                        ySnakeLength[0]=100;
```

```
            ySnakeLength[1]=100;
            ySnakeLength[2]=100;

        }


        //border of title image
        g.setColor(Color.white);
        g.drawRect(24,10,851,55);

        titleImage = new ImageIcon("snaketitle.jpg");
        titleImage.paintIcon(this, g, 25, 11);

        //border of gameplay
        g.setColor(Color.white);
        g.drawRect(24,74,851,577);
        g.setColor(Color.BLACK);
        g.fillRect(25, 75, 850, 575);

        //draw the scores
        g.setColor(Color.WHITE);
        g.setFont(new Font("arial",Font.PLAIN,14));
        g.drawString("Scores : "+score,780, 30);

        g.drawString("Length : "+snakeLength,780, 50);


        rightMouthIcon=new ImageIcon("rightmouth.png");

rightMouthIcon.paintIcon(this,g,xSnakeLength[0],ySnakeLength[0]);

        for(int a=0;a<snakeLength;a++)
        {
            if(a==0 && rightDirection)
            {
                rightMouthIcon=new ImageIcon("rightmouth.png");

rightMouthIcon.paintIcon(this,g,xSnakeLength[a],ySnakeLength[a]);
```

```
                }
                if(a==0 && leftDirection)
                {
                        leftMouthIcon=new ImageIcon("leftmouth.png");

        leftMouthIcon.paintIcon(this,g,xSnakeLength[a],ySnakeLength[a]);
                }
                if(a==0 && upDirection)
                {
                        upMouthIcon=new ImageIcon("upmouth.png");

        upMouthIcon.paintIcon(this,g,xSnakeLength[a],ySnakeLength[a]);
                }
                if(a==0 && downDirection)
                {
                        downMouthIcon=new ImageIcon("downmouth.png");

        downMouthIcon.paintIcon(this,g,xSnakeLength[a],ySnakeLength[a]);
                }

                if(a!=0)
                {
                        snakeIcon=new ImageIcon("snakeimage.png");

        snakeIcon.paintIcon(this,g,xSnakeLength[a],ySnakeLength[a]);
                }
        }

        enemyIcon = new ImageIcon("enemy.png");

        if(xEnemyPos[xEnemyPosIndex]==xSnakeLength[0] &&
yEnemyPos[yEnemyPosIndex]==ySnakeLength[0])
        {
                snakeLength++;
                score++;
                xEnemyPosIndex = random.nextInt(34);
                yEnemyPosIndex = random.nextInt(23);
        }
```

```
            enemyIcon.paintIcon(this, g, xEnemyPos[xEnemyPosIndex],
yEnemyPos[yEnemyPosIndex]);

            for(int b=1;b<snakeLength;b++)
            {
                  if(xSnakeLength[b]==xSnakeLength[0] &&
ySnakeLength[b]==ySnakeLength[0])
                  {
                        rightDirection=false;
                        leftDirection=false;
                        upDirection=false;
                        downDirection=false;

                        g.setColor(Color.WHITE);
                        g.setFont(new Font("arial",Font.BOLD,50));
                        g.drawString("Game Over!", 300, 300);

                        g.setFont(new Font("arial",Font.BOLD,20));
                        g.drawString("Space to RESTART", 350, 340);

                  }
            }

            g.dispose();
      }
      @Override
      public void actionPerformed(ActionEvent arg0) {

            if(rightDirection)
            {
                  for(int i=snakeLength-1;i>=0;i--)
                  {
                        ySnakeLength[i+1]=ySnakeLength[i];
                  }
                  for(int i=snakeLength;i>=0;i--)
                  {
                        if(i==0)
                        {
```

```
                xSnakeLength[i]=xSnakeLength[i]+25;
            }
            else
            {
                xSnakeLength[i]=xSnakeLength[i-1];
            }
            if(xSnakeLength[i]>850)
            {
                xSnakeLength[i]=25;
            }
        }

        repaint();
    }
    if(leftDirection)
    {
        for(int i=snakeLength-1;i>=0;i--)
        {
            ySnakeLength[i+1]=ySnakeLength[i];
        }
        for(int i=snakeLength;i>=0;i--)
        {
            if(i==0)
            {
                xSnakeLength[i]=xSnakeLength[i]-25;
            }
            else
            {
                xSnakeLength[i]=xSnakeLength[i-1];
            }
            if(xSnakeLength[i]<25)
            {
                xSnakeLength[i]=850;
            }
        }

        repaint();
    }
```

```
if(upDirection)
{
	for(int i=snakeLength-1;i>=0;i--)
	{
		xSnakeLength[i+1]=xSnakeLength[i];
	}
	for(int i=snakeLength;i>=0;i--)
	{
		if(i==0)
		{
			ySnakeLength[i]=ySnakeLength[i]-25;
		}
		else
		{
			ySnakeLength[i]=ySnakeLength[i-1];
		}
		if(ySnakeLength[i]<75)
		{
			ySnakeLength[i]=625;
		}
	}

	repaint();
}
if(downDirection)
{
	for(int i=snakeLength-1;i>=0;i--)
	{
		xSnakeLength[i+1]=xSnakeLength[i];
	}
	for(int i=snakeLength;i>=0;i--)
	{
		if(i==0)
		{
			ySnakeLength[i]=ySnakeLength[i]+25;
		}
		else
```

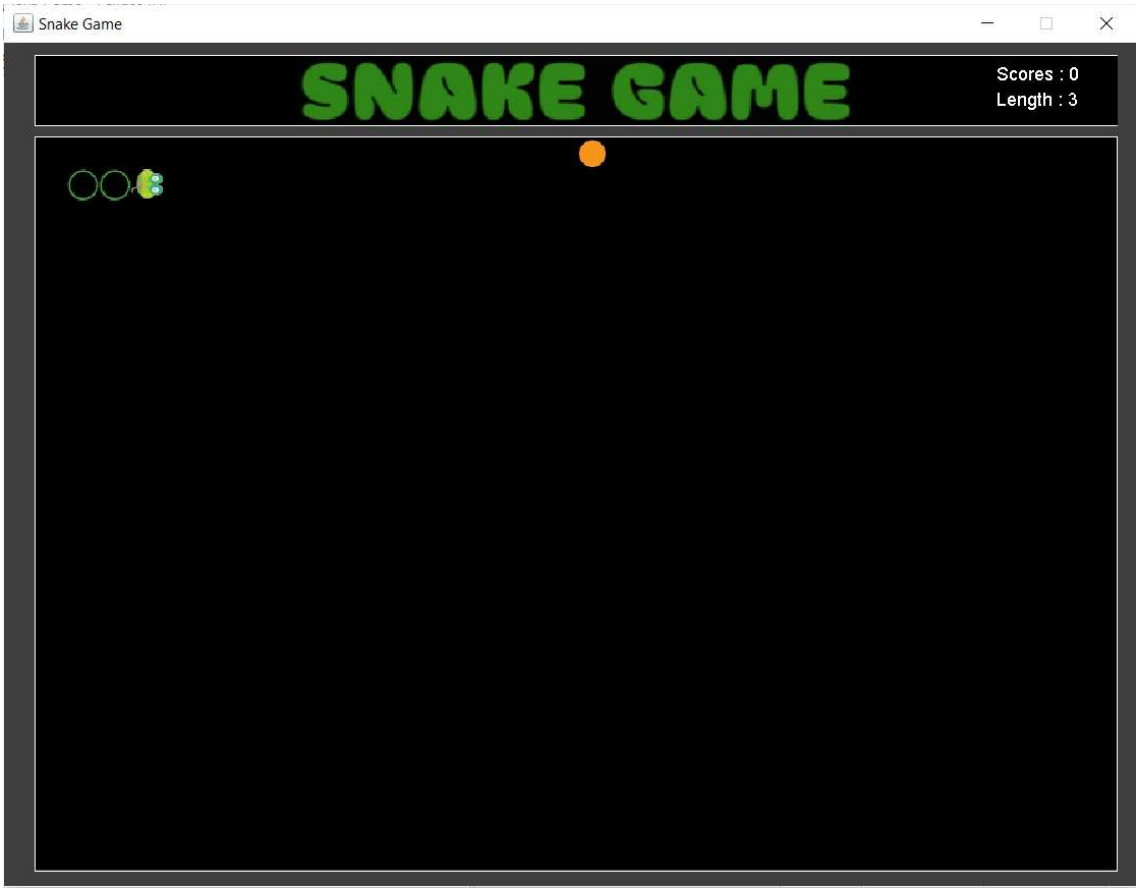
```
                {
                        ySnakeLength[i]=ySnakeLength[i-1];
                }
                if(ySnakeLength[i]>625)
                {
                        ySnakeLength[i]=75;
                }
        }

        repaint();
    }

}

@Override
public void keyPressed(KeyEvent e) {

    if(e.getKeyCode()==KeyEvent.VK_RIGHT)
    {
        moves++;
        if(!leftDirection)
        {
            rightDirection=true;
        }
        else
        {
            rightDirection=false;
            leftDirection=true;
        }
        upDirection=false;
        downDirection=false;
    }
    if(e.getKeyCode()==KeyEvent.VK_LEFT)
    {
        moves++;
        if(!rightDirection)
        {
            leftDirection=true;
```

```
        }
        else
        {
                leftDirection=false;
                rightDirection=true;
        }
        upDirection=false;
        downDirection=false;
}
if(e.getKeyCode()==KeyEvent.VK_UP)
{
        moves++;
        if(!downDirection)
        {
                upDirection=true;
        }
        else
        {
                upDirection=false;
                downDirection=true;
        }
        leftDirection=false;
        rightDirection=false;
}
if(e.getKeyCode()==KeyEvent.VK_DOWN)
{
        moves++;
        if(!upDirection)
        {
                downDirection=true;
        }
        else
        {
                downDirection=false;
                upDirection=true;
        }
        leftDirection=false;
        rightDirection=false;
```

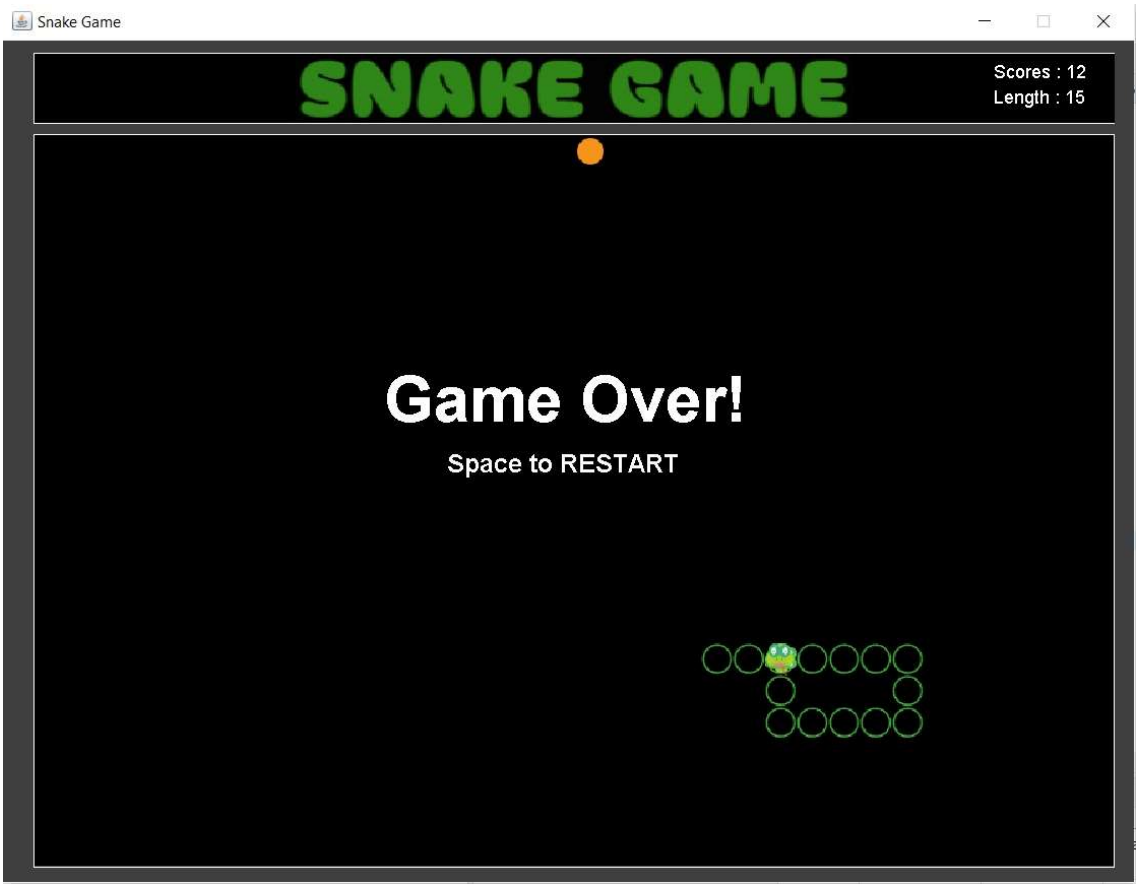
```
            }

            if(e.getKeyCode()==KeyEvent.VK_SPACE)
            {
                  score=0;
                  moves=0;
                  snakeLength=3;
                  repaint();
            }

      }

      @Override
      public void keyReleased(KeyEvent arg0) {
            // TODO Auto-generated method stub

      }
      @Override
      public void keyTyped(KeyEvent arg0) {
            // TODO Auto-generated method stub

      }


}
```

Output and Screenshots:

This is the starting game window of the snake game.



This is output after game over condition are met.



Conclusion

In conclusion, the Snake Game project provides a simple yet entertaining implementation of the classic arcade game using Java and Swing. It demonstrates fundamental concepts such as user input handling, animation, collision detection, and graphical rendering. Further enhancements could include additional features such as levels, high scores, and improved graphics to make the game more engaging.

Future Scope

While the Snake Game project already provides a satisfying gaming experience, there are several avenues for future enhancements and improvements:

- High Scores and Leaderboards : Incorporate a high-score system to track players' performance over multiple sessions. Display a leaderboard to showcase top scores and encourage competition among players.

- Improved Graphics and Animations : Enhance the visual appeal of the game with more detailed graphics, animations, and visual effects. Utilize advanced graphics libraries or frameworks to create a more immersive gaming environment.

- Sound Effects and Music : Add sound effects for actions such as snake movement, eating food, and game over events. Include background music or customizable soundtracks to enhance the overall atmosphere of the game.

- Mobile Compatibility : Adapt the game for mobile platforms by optimizing the user interface and controls for touchscreen devices. Publish the game on app stores to reach a wider audience and enable on-the-go gaming.

- Multiplayer mode: Local multiplayer mode as well as online multiplayer mode can be made so that more than one player can have match against each other.

- Advanced gameplay features: Level and changes, mission or task based gameplay can be made to make it more attractive.