SNAKE GAME

A Course Based Project Submitted in Partial Fulfilment of the Requirement for the Award of the degree of

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING - ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Submitted by

21071A7230 – K. Ravi Chandra

21071A7250 – P. Nitin

21071A7252 – P. Srijith

DEPARTMENT OF CSE-CYS, DS & (AI &DS)

VALLURUPALLI NAGESWARARAO VIGNANA JYOTHI

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute, NAAC Accredited With 'A++' Grade, NBA Accredited, Approved by AICTE, New Delhi, Affiliated to JNTUH)

VALLURUPALLI NAGESWARARAO VIGNANA JYOTHI

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute)

CERTIFICATE

This is to Certify K. Ravi Chandra (21071A7230), P. Nitin (21071A7250) and P. Srijith(21071A7252) that they have successfully completed their project work at CSE CYS, DS & (AI & DS) Department of VNRVJIET, Hyderabad entitled “Snake Game” in partial fulfilment of the requirements for the award of the Bachelor of Technology degree during the Academic year 2022-2023.

Project Guide	Head of Department
Mrs. E. Lalitha	Dr. Mr. Rajasekhar
Assistant Prof. and Internal Guide	Prof. and Head
Dept. of CSE-CYS, DS and AI&DS	Dept. of CSE-CYS, DS and AI&DS
VNRVJIET	VNRVJIET

DECLARATION

This is to certify that the project work entitled "Snake Game" submitted in VNR Vignana Jyothi Institute of Engineering & Technology in partial fulfilment of requirement for the award of Bachelor of Technology in Computer Science and Engineering. It is a Bonafide report of the work carried out by us under the guidance and supervision of Mrs. E. Lalitha (Assistant Professor), Department of CSE-CYS, DS, AI&DS, VNRVJIET. To the best of our knowledge, this report has not been submitted in any form to any university or institution for the award of any degree or diploma.

K. Ravi Chandra	P. Nitin	P. Srijith
(21071A7230)	(21071A7250)	(21071A7252)
CSE - AI & DS	CSE - AI & DS	CSE - AI & DS

ACKNOWLEDGEMENT

Behind every achievement lies the heartfelt gratitude to those who activated in completing the project. To them we lay the words of gratitude within us.

We are indebted to our venerable principal Dr. C. D. NAIDU for this inflicting devotion, which led us to complete this project. The support, encouragement given by him and his motivation led us to complete the project.

We express our sincere thanks to internal guide Mrs. E. Lalitha and also Head of the Department Dr. M. RAJA SHEKHAR for having provided us a lot of facilities to undertake the project work and guide us to complete the project.

We take the opportunity to express thanks to our faculty of the Dept. of COMPUTER SCIENCE AND ENGINEERING – ARITIFICAL INTELLIGENCE AND DATA SCIENCE and remaining members of our college VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY who extended their valuable support in helping us to complete the project in time.

K. Ravi Chandra (21071A7230)

P. Nitin (21071A7250)

P. Srijith
(21071A7252)

ABSTRACT

The Snake game is a classic arcade game that has been around for decades. It is a simple game where the player controls a snake that moves around the screen and eats food, growing in length as it goes. The game is won by reaching a certain length or score, or lost when the snake collides with the walls or itself. A Snake game designed using Java GUI can provide an interactive and user-friendly experience for players. The Java GUI allows for the creation of a graphical interface with buttons, graphics, and other visual elements that can enhance the overall look and feel of the game. The game logic can be implemented using Java's object-oriented programming features, making it easier to create a well-structured and organized code. The game can be designed with different levels of difficulty and various game modes, providing a challenging and engaging experience for players. In conclusion, a Snake game designed using Java GUI can provide a fun and entertaining experience for players while also showcasing the capabilities of Java as a programming language. With its simple gameplay mechanics and customizable features, the Snake game remains a popular choice for game developers and players alike.

CONTENTS

INTRODUCTION

Java Snake Game is a classic arcade-style game that has been enjoyed by generations of gamers. The objective of the game is to control a snake that moves around a board, eating food and growing longer with each bite. The player must avoid running into the walls or the snake's own body, which ends the game. The game is simple to play but difficult to master, providing hours of entertainment.

The Java Snake Game was originally developed by Nokia in the 1990s as a pre-installed game on their mobile phones. The game quickly became popular and has since been adapted for various platforms, including computers and other mobile devices.

The game is written in the Java programming language, which is a popular language used for developing various applications, including games. The Java Snake Game code is open source and freely available, which has led to many variations of the game being developed by the gaming community.

The Java Snake Game is a great way for beginners to learn programming, as it provides a practical application of programming concepts such as data structures, loops, and functions. It is also a great way to develop problem-solving skills and logical thinking. Overall, the Java Snake Game is a fun and challenging game that has stood the test of time and continues to be enjoyed by gamers of all ages.

LIBRARIES

Java Swing is a graphical user interface (GUI) toolkit that is part of the Java Standard Edition (Java SE) platform. It is used to create desktop applications with a user-friendly interface that allows users to interact with the application. Swing is built on top of the Abstract Window Toolkit (AWT) and provides a set of components, such as buttons, text fields, and menus, that can be used to build graphical user interfaces.

The Swing library provides a wide range of features and tools to developers, including the ability to customize the look and feel of the GUI, internationalization and localization support, accessibility features, and more. It also provides a set of layout managers that allow developers to control the positioning and sizing of components within a container.

One of the main advantages of using Swing is that it is platform-independent and can be used on any operating system that supports Java. This allows developers to write once and run anywhere, without having to worry about the specific details of each operating system.

Overall, the Swing library is a powerful and flexible tool for building desktop applications with a user-friendly interface. It is widely used in the development of Java applications and provides developers with a range of tools and features to create high-quality applications.

CODE FOR SNAKE GAME

Code in SnakeGame.java

```
public class SnakeGame {
	public static void main(String[] args) {
		new GameFrame();
	}
}
```

Code in GameFrame.java

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
public class GameFrame extends JFrame {
	GameFrame() {
		this.add(new GamePanel());
		ImageIcon image = new ImageIcon("snake3.png");
		this.setIconImage(image.getImage());
		this.setTitle("Snake Game");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setResizable(false);
		this.pack();
		this.setVisible(true);
		this.setLocationRelativeTo(null);
	}
}
```

Code in GamePanel.java

```
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 = 600;
	static final int SCREEN_HEIGHT = 600;
	static final int UNIT_SIZE = 25;
	static final int GAME_UNITS = 
(SCREEN_WIDTH*SCREEN_HEIGHT)/UNIT_SIZE;
```

```
	static final int DELAY = 75;
	final int x[] = new int[GAME_UNITS];
	final int y[] = new int[GAME_UNITS];
	int bodyparts = 6;
	int applesEaten;
	int appleX, 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) {
			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(40,180,0));
					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,30));
                        FontMetrics metrics = getFontMetrics(g.getFont());
                        g.drawString("Score: "+applesEaten, (SCREEN_WIDTH -
metrics.stringWidth("Score: "+applesEaten)), 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 checkCollosions() {
		//if head collides body
		for(int i=bodyparts;i>0;i--) {
			if((x[0]==x[i]) && (y[0]==y[i])) {
				running = false;
			}
		}
		if(x[0]<0 || x[0]>=SCREEN_WIDTH) {
			running = false;
		}
		if(y[0]<0 || y[0]>=SCREEN_HEIGHT) {
			running = false;
		}
		if(running == false) {
			timer.stop();
		}
	}
	public void gameOver(Graphics g) {
		g.setColor(Color.green);
		g.setFont(new Font("Ink Free",Font.BOLD,50));
		FontMetrics metrics1 = getFontMetrics(g.getFont());
		g.drawString("Score: "+applesEaten, (SCREEN_WIDTH - 
metrics1.stringWidth("Score: "+applesEaten))/2, g.getFont().getSize());
		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();
			checkCollosions();
		}
		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;
                        }
                }
        }
}
```

Description of the Code:

This code implements a simple version of the game Snake using Java and Swing library. The game consists of a snake that moves around the screen eating apples, and the player's goal is to eat as many apples as possible without crashing into the snake's body or the screen's edges.

The `SnakeGame` class has the `main` method that starts the game by creating a new instance of `GameFrame`.

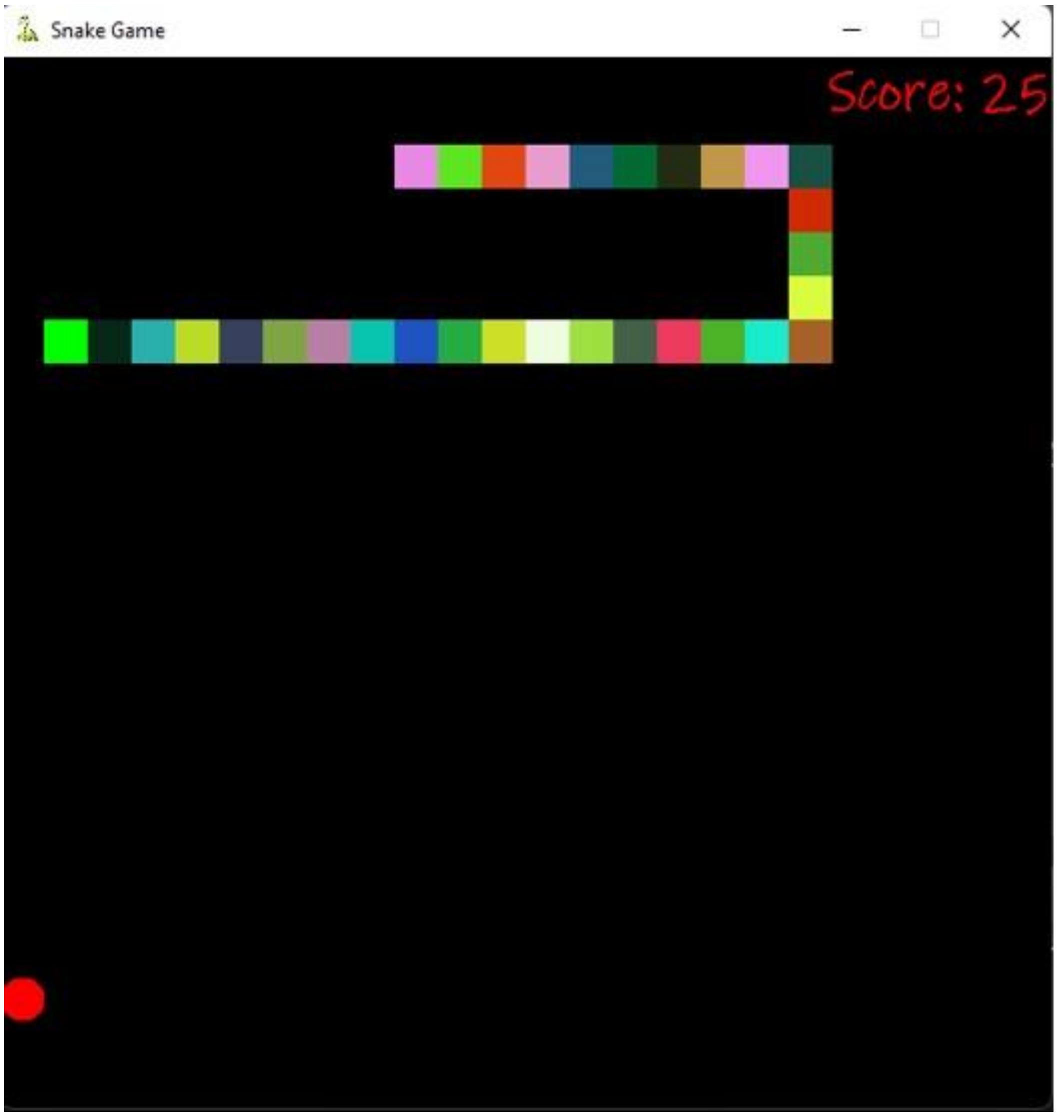
The `GameFrame` class extends `JFrame` and sets up the main window of the game. It creates a new instance of `GamePanel` and adds it to the frame, sets the title, icon, size, and visibility of the window, and centers it on the screen.

The `GamePanel` class extends `JPanel` and is the main component of the game. It sets up the game environment, handles the user input, updates the game state, and renders the game objects. It has a `startGame` method that initializes the game variables and starts the game loop with a fixed delay of 75 milliseconds. It also has methods for drawing the apple, the snake's body, and the score on the screen, and for handling collisions and game over conditions. Finally, it implements the `ActionListener` interface to handle the game loop and the `KeyListener` interface to handle the arrow keys pressed by the player.

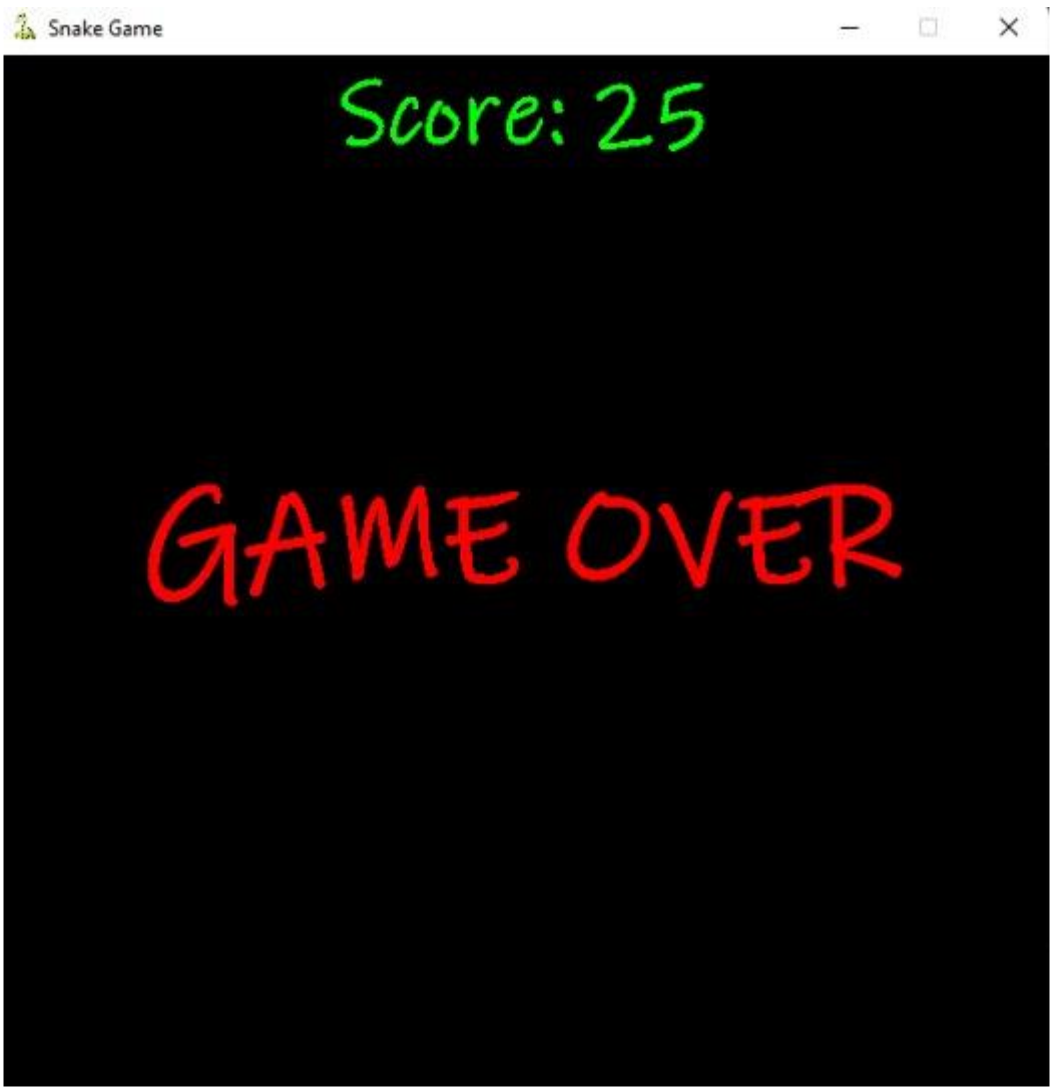
The game uses a simple algorithm to move the snake's body by shifting each body part one position ahead and updating the head's position based on the current direction. The game checks for collisions with the apple, the snake's body, and the screen edges by comparing the head's position with the other objects. If the snake eats an apple, it grows by one body part and the score increases. If the snake collides with its body or the screen edges, the game ends, and the score is displayed on the screen.

OUTPUT

When playing:



When Defeated:



CONCLUSION

The code is an implementation of the classic Snake game in Java, using Swing for the graphical user interface. The game consists of a snake that moves around the screen and eats apples to grow in size. The player must avoid colliding with the walls or with the snake's own body. The game keeps track of the player's score, which increases as more apples are eaten. The code is divided into three classes: SnakeGame, GameFrame, and GamePanel. The SnakeGame class contains the main method, which creates a new GameFrame object. The GameFrame class extends JFrame and sets up the window for the game. The GamePanel class extends JPanel and contains the game logic and rendering code. Overall, the code is well-structured and easy to follow, with clear separation of concerns between the different classes.