B. Tech Program First Year

Experiential Learning

DA1001

## “SNAKE GAME”

by

**Srishti Suman**

(Reg. No: 209303160)

**Shreya Avadhuta**

(Reg. No: 209303233)

**Tanuj Kothari**

(Reg. No: 209303227)

Under the guidance

of

**Dr Kusum Lata Jain**

**Assistant Professor**

Computers & Communication Engineering, School of Computing & Information Technology, Manipal University Jaipur, Jaipur (Raj.).

**Department of Computers & Communication Engineering**

**School of Computing & Information Technology**

**Faculty of Engineering**

**Manipal University Jaipur, India**

December 2020

# Certificate

This is to certify that the project titled **“Snake Game”** is a record of the bona fide work done by **Srishti Suman** (Reg. No: 209303160), **Shreya Avadhuta** (Reg. No: 209303233) & **Tanuj Kothari** (Reg. No: 209303227) submitted for the partial fulfilment of the requirements for the completion of the Experiential Learning (DA1001) course in the Department of Computers & Communication Engineering of Manipal University Jaipur, during the academic session July-November 2020.

|  |
| --- |
| *Signature of the mentor* |
| Dr Kusum Lata Jain  Assistant Professor  Department of Computers & Communication Engineering |
| *Signature of the HoD* |
| Dr Vijaypal Singh Dhaka  Head of the Department  Department of Computers & Communication Engineering |

# Abstract

# In this project, we have re-created the iconic Snake Game. The goal of this game is to make the snake eat the tokens which are its food without the snake crashing to its own body. Every time the snake eats the token the score is updated. The player loses when the snake eats/crashes into its own body. Object Oriented Programming was the main principle followed in the project. The project has been created using the Java language, in Visual Studio Code, in the OpenJDK 15 environment by RedHat Developer.

# Introduction

# Background Of Java:

# Developed and created by John Gosling in 1995 in Sun Microsystems, Java is a general-purpose, object-oriented programming language. It was developed and intended to follow the WORA concept which means Write Once Run Anywhere i.e., compiled Java code can run on all platforms that support Java without the need for recompilation.

# Java offers various applications in mobile development, web application development being the major areas. Apart from this, it has applications in Desktop applications, Web servers and application servers, games, database connection. It also offers its support in embedded systems and scientific applications.

# Reasons for choosing Java:

# Simple

# Object-Oriented

# Portable

# Platform independent

* High Performance
* Dynamic

The project was to re-create the Sanke Game to build skill in the use of Digital Systems, Object Oriented Programming design and implementation using classes and objects, as well as designing and using multiple classes.

**Defining The Task:**

To create a Snake Game that allows users to control the movement of a snake on the screen, to get points for eating food and avoid running into the growing tail of the snake itself.

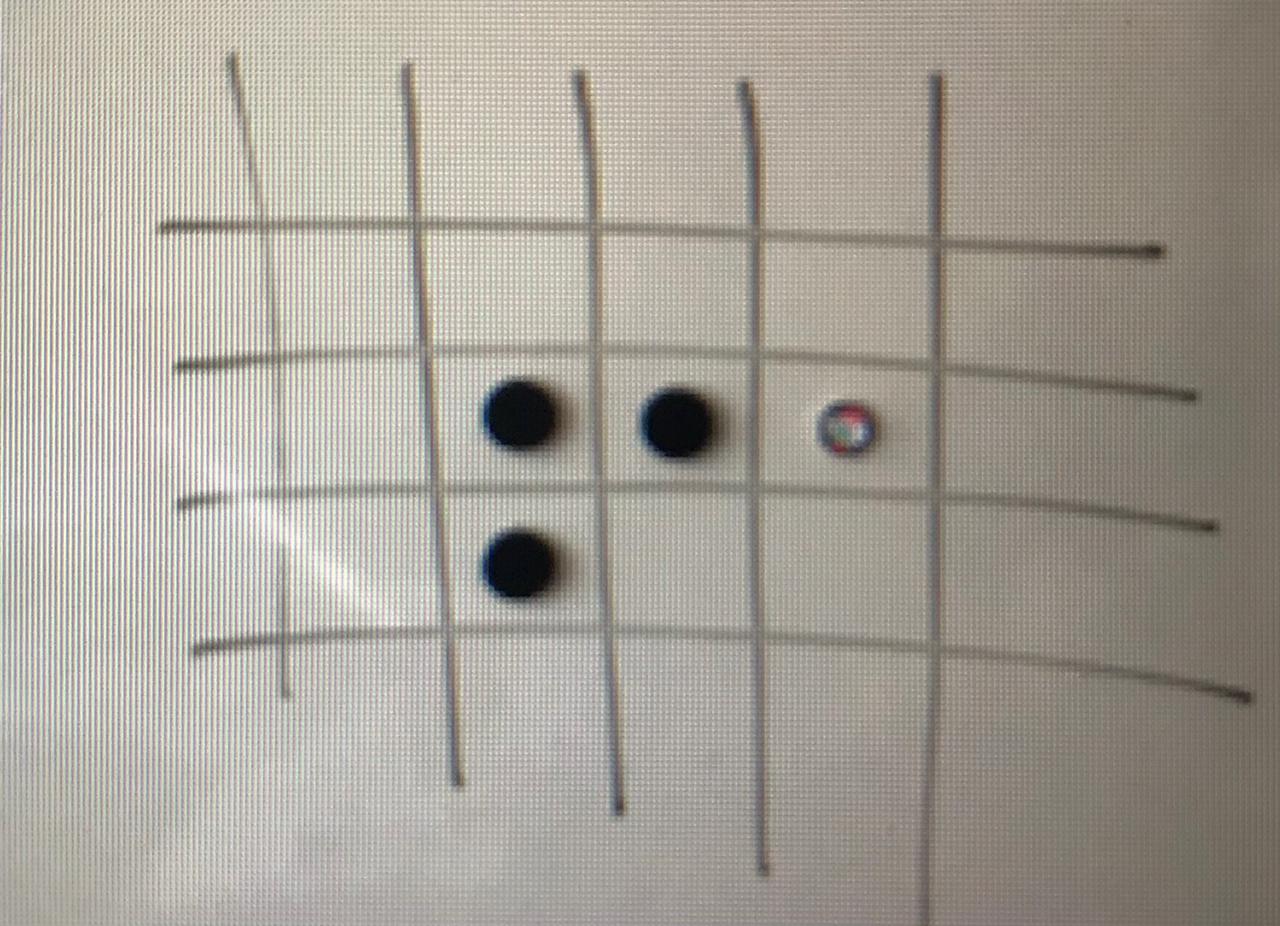
For this, we want to write a game where a graphical representation of a snake moves across the screen. When it encounters a piece of food, the snake grows longer, and we gain a point. If it runs into itself, we die. The snake is controlled by our keyboard.

To make this, we require:

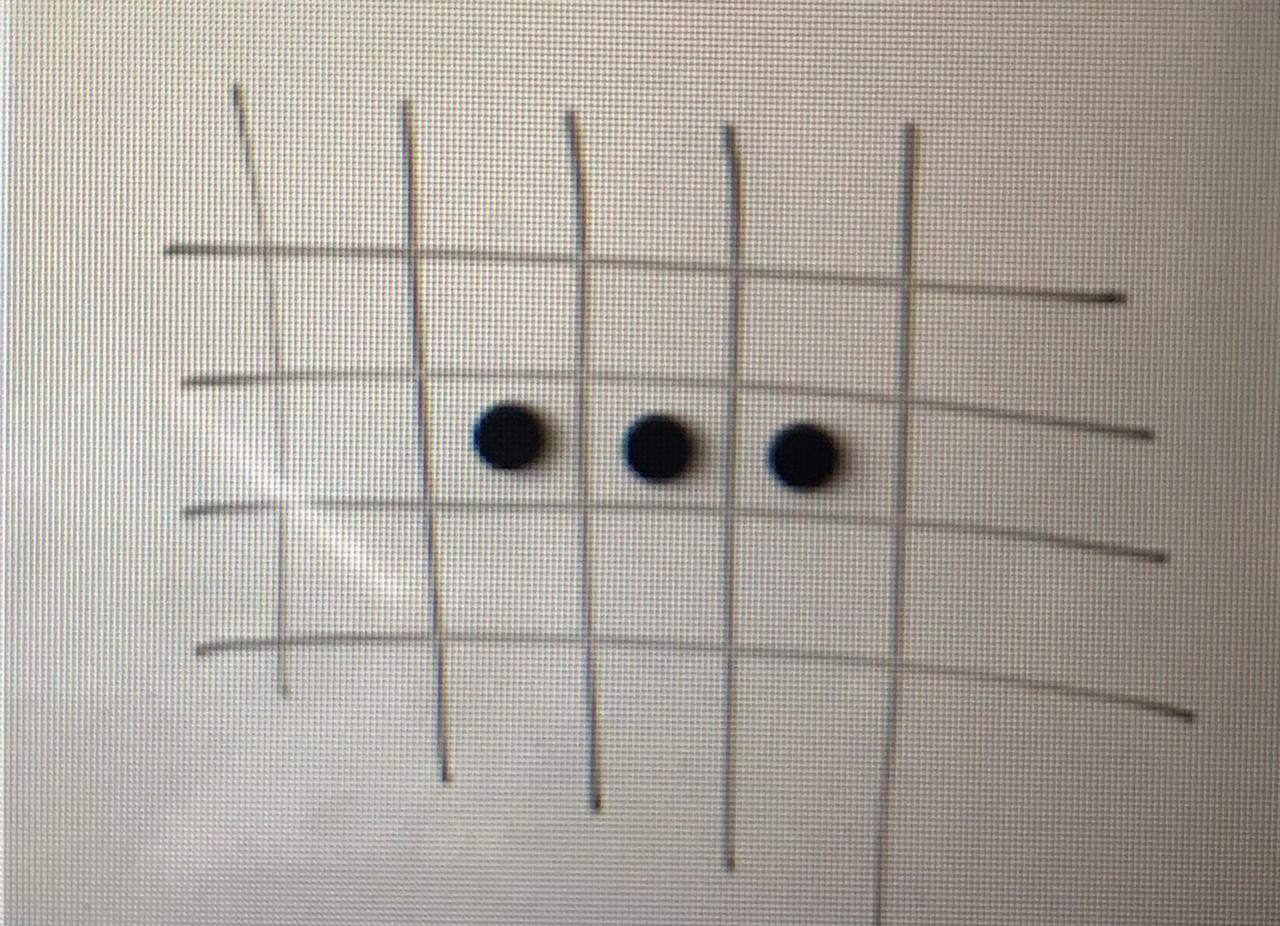
* A way of representing the snake
* A way of representing the food
* A way for our instructions to reach the snake
* A way to know when the snake bites its own body/game end
* A file controlling all game logic and threads

**Prototyping and UI Design:**

Sketch prototypes are very useful to get a feel for how the game will work. Here are the examples, showing a snake moving, eating food, and then growing after eating.



Here the snake is approaching the food from the right.



The snake moves to enter the spot, with the last elements of its tail being dragged along, as the snake is only three units long.

A picture containing indoor

Description automatically generated

But now it has eaten, so it increases in length, and we have steered it up.

**Java Functions & Extensions Imported:**

* **java.util.ArrayList**: Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list (This class is roughly equivalent to Vector, except that it is unsynchronized).
* **java.awt.Color:** It is used to encapsulate colours in the default sRGB colour space or colours in arbitrary colour spaces identified by a ColorSpace.
* **java.awt.event.KeyAdapter:** An abstract adapter class for receiving keyboard events. The methods in this class are empty. This class exists as convenience for creating listener objects.
* **java.awt.event.KeyEvent:** An event which indicates that a keystroke occurred in a component.
* **javax.swing.JPanel:** JPanel is a generic lightweight container.
* **javax.swing.JFrame:** An extended version of java.awt.Frame that adds support for the JFC/Swing component architecture.
* **java.util.ArrayList:** Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list.
* **java.awt.GridLayout:** The GridLayout class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.
* **java.awt.event.KeyListener:** The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event either implements this interface (and all the methods it contains) or extends the abstract KeyAdapter class (overriding only the methods of interest).

**Additional Notes:**

We utilize various predefined Java functions and extensions to create the project.

Students work in teams to design and implement the Java program. The project is well suited to teamwork and can be used to build collaboration and communication skills.

The activity is about experimenting, trying new solutions, and debugging. It is a very simple game, but the program developed has a development process found in real-world scenarios.

# Methodology

The game logic:

1. Draw the playing area, set the counter to 0 and display it.
2. Draw the snake in starting position
3. Draw the food in starting position
4. On user input, change snake direction
5. If the snake is over food, eat it, increase the tab, grow, and shift the food
6. Else if the snake is over itself, die
7. Go back to 4
8. Repeat until the snake dies

The full Java code for the project is linked below. It is automatically connected to a **.jar** file which runs the game. The description/synopsis of all the classes is as follows:

**Class DataOfSquare:**

This imports java.awt.Color function and houses the array list that contains the colours for all the elements present.

**Class KeyboardListener:**

This imports java.awt.event.KeyAdapter & java.awt.event.KeyEvent functions and contains the instructions for accepting input from the keyboard and performing the required task.

**Class SquarePanel:**

This imports java.awt.Color & javax.swing.JPanel functions and houses the data about the game panel base.

**Class Tuple:**

This is the class which contains the coordinate data of all the assets of the game.

**Class Window:**

This imports java.awt.GridLayout, java.awt.event.KeyListener, java.util.ArrayList & javax.swing.JFrame functions. It creates the array list that contains the threads, creates threads and its data, and adds it to the array list. It sets up the layout of the panel, and starts & pauses all threads, then adds every square of each thread to the panel. It houses the initial position of the snake, passes the value to the controller, and starts the game. It links to the window to the KeyboardListener class mentioned above.

**Class ThreadsController:**

This is the most important class in the project and controls all game logic. It imports java.util.ArrayList function. The details about all the sections for this program are mentioned in the file itself alongside the code itself as comment lines.

**Class Main:**

The main class for the project. It imports javax.swing.JFrame and starts the window for the **.jar** game file.

# Results and Discussions

**Software Used To Obtain The Result:**

* Programming language: Java
* Code Editor: Visual Studio Code
* Development Environment: OpenJDK 15 by RedHat

**Understanding The Data:**

Most of the data representation in the program is straightforward. The position of the food is given by an (x,y) coordinate pair but the food’s move function needs to know the size of the canvas if it is going to randomly appear in a place where we can eat it.

**Testing & Evaluation:**

How can we test a full Snake Game and, assuming it passes that stage, how can we playtest that?

The functional requirements can be turned into a checklist. Let us look at an example: To display the snake, the first thing we want to do is to make sure that we can draw the snake and move it around on the screen. So, our testing for correct function will be:

* Is the snake's head displayed on the screen?
* Is it moving around properly using keyboard control?
* Is it displaying correctly?
* Is the body moving correctly?

**A Major Part Of The Source Code:**

import java.util.ArrayList;

//Controls all the game logic - most important class in this project.

public class ThreadsController extends Thread

{

    ArrayList<ArrayList<DataOfSquare>> Squares= new ArrayList<ArrayList<DataOfSquare>>();

    Tuple headSnakePos;

    int sizeSnake=3;

    long speed = 50;

    public static int directionSnake;

    ArrayList<Tuple> positions = new ArrayList<Tuple>();

    Tuple foodPosition;

    //Constructor of ControllerThread

    ThreadsController(Tuple positionDepart)

    {

        //Get all the threads

        Squares=Window.Grid;

        headSnakePos=new Tuple(positionDepart.x,positionDepart.y);

        directionSnake = 1;

        //Pointer

        Tuple headPos = new Tuple(headSnakePos.getX(),headSnakePos.getY());

        positions.add(headPos);

        foodPosition= new Tuple(Window.height-1,Window.width-1);

        spawnFood(foodPosition);

    }

    //Important part :

    public void run()

    {

        while(true)

        {

            moveInterne(directionSnake);

            checkCollision();

            moveExterne();

            deleteTail();

            pauser();

        }

    }

    //delay between each move of the snake

    private void pauser()

    {

        try

        {

            sleep(speed);

        }

        catch (InterruptedException e)

        {

            e.printStackTrace();

        }

    }

    //Checking if the snake bites itself or is eating

    private void checkCollision()

    {

        Tuple posCritique = positions.get(positions.size()-1);

        for(int i = 0;i<=positions.size()-2;i++)

        {

            boolean biteItself = posCritique.getX()==positions.get(i).getX() && posCritique.getY()==positions.get(i).getY();

            if(biteItself)

            {

                stopTheGame();

            }

        }

        boolean eatingFood = posCritique.getX()==foodPosition.y && posCritique.getY()==foodPosition.x;

        if(eatingFood)

        {

            System.out.println("Yummy!");

            sizeSnake=sizeSnake+1;

            foodPosition = getValAleaNotInSnake();

            spawnFood(foodPosition);

        }

    }

    //Stops The Game

    private void stopTheGame()

    {

        System.out.println("COLISION! \n");

        while(true)

        {

            pauser();

        }

    }

    //Put food in a position and displays it

    private void spawnFood(Tuple foodPositionIn)

    {

        Squares.get(foodPositionIn.x).get(foodPositionIn.y).lightMeUp(1);

    }

    //return a position not occupied by the snake

    private Tuple getValAleaNotInSnake()

    {

        Tuple p ;

        int ranX= 0 + (int)(Math.random()\*19);

        int ranY= 0 + (int)(Math.random()\*19);

        p=new Tuple(ranX,ranY);

        for(int i = 0;i<=positions.size()-1;i++)

        {

            if(p.getY()==positions.get(i).getX() && p.getX()==positions.get(i).getY())

            {

                ranX= 0 + (int)(Math.random()\*19);

                ranY= 0 + (int)(Math.random()\*19);

                p=new Tuple(ranX,ranY);

                i=0;

            }

        }

        return p;

    }

    //Moves the head of the snake and refreshes the positions in the arraylist

    //1:right 2:left 3:top 4:bottom 0:nothing

    private void moveInterne(int dir)

    {

        switch(dir)

        {

            case 4:

                headSnakePos.ChangeData(headSnakePos.x,(headSnakePos.y+1)%20);

                positions.add(new Tuple(headSnakePos.x,headSnakePos.y));

                break;

            case 3:

                if(headSnakePos.y-1<0)

                {

                    headSnakePos.ChangeData(headSnakePos.x,19);

                }

                else

                {

                    headSnakePos.ChangeData(headSnakePos.x,Math.abs(headSnakePos.y-1)%20);

                }

                positions.add(new Tuple(headSnakePos.x,headSnakePos.y));

                break;

            case 2:

                if(headSnakePos.x-1<0)

                {

                    headSnakePos.ChangeData(19,headSnakePos.y);

                }

                else

                {

                    headSnakePos.ChangeData(Math.abs(headSnakePos.x-1)%20,headSnakePos.y);

                }

                positions.add(new Tuple(headSnakePos.x,headSnakePos.y));

                break;

            case 1:

                headSnakePos.ChangeData(Math.abs(headSnakePos.x+1)%20,headSnakePos.y);

                positions.add(new Tuple(headSnakePos.x,headSnakePos.y));

                break;

        }

    }

    //Refresh the squares that needs to be

    private void moveExterne()

    {

        for(Tuple t : positions)

        {

            int y = t.getX();

            int x = t.getY();

            Squares.get(x).get(y).lightMeUp(0);

        }

    }

    //Refreshes the tail of the snake, by removing the superfluous data in positions arraylist

    //and refreshing the display of the things that is removed

    private void deleteTail()

    {

        int cmpt = sizeSnake;

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

        {

            if(cmpt==0)

            {

                Tuple t = positions.get(i);

                Squares.get(t.y).get(t.x).lightMeUp(2);

            }

            else

            {

                cmpt--;

            }

        }

        cmpt = sizeSnake;

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

        {

            if(cmpt==0)

            {

                positions.remove(i);

            }

            else

            {

                cmpt--;

            }

        }

    }

}

import javax.swing.JFrame;

public class Main

{

    public static void main(String[] args)

    {

        Window f1= new Window();

        //Window settings

        f1.setTitle("Snake");

        f1.setSize(500,500);

        f1.setVisible(true);

        f1.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

}

**The Output:**

The various stages of the game in progress are displayed in the images below:

Text

Description automatically generated

The VS Code application window before the running the **.jar** file to start the game.

A screenshot of a computer

Description automatically generated

Start of the game with the snake and the food in their initial positions. The terminal window does not display anything yet, as the snake has not consumed any food.

A screenshot of a computer

Description automatically generated

The game window after a significant amount of progress. Notice the terminal window changing to indicate the quantity of food the snake has consumed up to this point.

A screenshot of a computer

Description automatically generated

The end of the game as the snake collides with its own body. The terminal window now displays the message “COLISION!” to signify the crash.

**Other Discussion Points:**

To keep the game interesting,

* The snake must move at a speed fast enough to be interesting but slow enough to control.
* The game should still be playable with a long snake.

# Future Prospects

# There are many ways this could be extended. Some that we have thought of include:

# How could we go about producing a two-player version of the game?

# Try to make the snake speed up as it gets longer.

# Try to add controller support.

# Try to use a timer to make the snake starve if it does not eat quickly enough.

# References

The following websites were used to get inspiration for the project, and to do some background research:

1. <https://hackr.io/blog/java-projects>
2. <https://www.upgrad.com/blog/java-project-ideas-topics-for-beginners/>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_