B.M.S COLLEGE OF ENGINEERING

P.O. Box No.: 1908 Bull Temple Road,

Bangalore - 560019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

JAVA PROGRAMMING

19IS4PCJAV

Snake Game

Submitted by:

1BM19IS134- PAVITHRA S

1BM18IS125- YASHAS S REDDY

1BM19IS102- NIKHIL V WADHWA

Submitted to:

Mrs. Sindhu K

Assistant Professor

B.M.S COLLEGE OF ENGINEERING

P.O. Box No.: 1908 Bull Temple Road,

Bangalore - 560019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that the Project has been successfully presented at **B.M.S College Of Engineering** by **PAVITHRA S, YASHAS S REDDY, NIKHIL V WADHWA** bearing USN: **1BM19IS134**, **1BM18IS125**, and **1BM19IS102** in partial fulfillment of the requirements for the IV Semester degree in **Bachelor of Engineering in Information Science & Engineering** of **BMS COLLEGE OF ENGINEERING Affiliated By Visvesvaraya Technological University, Belgaum** as a part of the course **JAVA Programming *(*19IS4PCJAV)** during the academic year 2020-2021.

Faculty Name: Sindhu K

Designation: Assistant Professor

Department of ISE, BMSCE

TABLE OF CONTENTS

SNO	CONTENT	PAGE NO
1	**ABSTRACT**	4
2	**INTRODUCTION**	5
3	**OVERVIEW OF THE PROJECT**	7
4	**Rules of the game**	4
5	**CONCEPTS APPLIED**	9
6	**IMPLEMENTATION**	9
7	**CODE**	13
8	**SNAPSHOT**	26
9	**References**	27

ABSTRACT

Snake is an older classic video game. It was first created in the late 70s. Later it was brought to PCs. In this game, the player controls a snake. The objective is to eat as many apples as possible. Each time the snake eats an apple its body grows. The snake must avoid the walls and its own body. This game is sometimes called Nibbles.

Development of Java Snake game

The size of each of the joints of a snake is 10 px. The snake is controlled with the cursor keys. Initially, the snake has three joints. If the game is finished, the "Game Over" message is displayed in the middle of the board.

Problem Statement

The problem is to design a Snake Game which provides the following functionalities:

- Snakes can move in a given direction and when it eats the food, the length of snake increases.
- When the snake crosses itself, the game will be over.
- Food will be generated at a given interval.

The main classes that can be used are:

1. Snake

2. Board

INTRODUCTION

This programming assignment is a simplified version of the classic game Snake. For those that are not familiar with the game Snake, it is a game where you are a snake and your goal is to move around and eat pieces of food (in our case, Fruit Loops). Once the game starts, the snake moves continuously in the direction you tell it to. Each time the snake eats a piece of food (a Fruit Loop), it grows. The end goal is to eat enough food such that the snake covers the entire game area. You lose the game if the snake runs into itself or the borders of the game area (as you can imagine, this becomes more difficult as the snake fills up more of the game area).

There are 4 main components to the implementation of this version of Snake: the grid, the snake, the fruit loops, and the controller. The snake will move around on the grid, with the goal to eat fruit loops placed randomly on the grid. In this version of the game, the game is won once the snake eats a specified number of fruit loops. The controller will be responsible for taking in user input through the keyboard (where arrow keys will be used to move the snake) and directing the snake on what to do based on the input.

OVERVIEW OF THE PROJECT

Movement

Determining the movement of the snake is the largest part of our code. The first thing that is done is to determine if the user wants to change directions. This is done by comparing the values from the ADC to find out if one is high or low. Then if a direction is indicated it is evaluated to see if this is a valid change in direction based on the user's current direction. For instance, the user can't go down if they are going up and they are not changing directions if the joystick is held up. Then this information is stored and if the user is changing directions there direction is changed and stored using the struct explained in setup. For efficiency only, the last and first segments of the snake are drawn this prevents the screen from lagging.

Collision detection

We use the game board described above for collision detection. Player 1 is denoted by A "1" if a snake runs into either of these on the game board then they have hit something and that The snake's movement is done. An apple is denoted by an "X" which is how we determine if a sound is played and if a new fruit is spawned onto the screen. A "-" indicates that we are not colliding with anything while any other value indicates we have collided with something and The snake is declared dead using the variable hit. Hit is part of all of the movement comparisons and must be false for the snake to move.

Fruit Randomization

For fruit randomization, there is a variable called “count”. This variable is constantly incremented in the getGameType, getNumPlayers, and get difficulty as well as the main loop. What this means is from when you turn on the game console we will have an effective random number because the odds a user can select input the same every time is astronomical. The placement of the fruit is also determined by how long it takes the snake to eat another fruit. The random fruit generator takes in the count value and mods it by the height and width of the screen plus one and gets an effective coverage of the entire screen. Some things that had to be taken into consideration were the fact that the program cannot spawn a fruit in a location that doesn’t have a “-“ on the game board if it does happen to choose that spot then the program will add a one to that count to tell it finds an empty location. We decided to do this because on The microcontroller seemed to be the most effective way to get a totally random number.

Rules of the game

The snake starts at the center of the board, moving north (upward).

The snake moves at a constant speed.

The snake moves only north, south, east, or west (ignore the versions of the game where the snake can move in curves).

The snake "moves" by adding dots to its head and simultaneously deleting a dot from the tip of its tail.

"Apples" appear at random locations, and persist for a random amount of time (but usually long enough for it to be possible for the snake to get to the apple).

There is always exactly one apple visible at any given time.

When the snake "eats" (runs into) an apple, it gets longer.

(This is hard to describe, so play a couple of games to see what I mean.) When the snake gets longer, say by n dots, it does so by not deleting dots from its tail for the next n moves.

The game continues until the snake "dies".

A snake dies by either (1) running into the edge of the board, or (2) by running into its own tail.

The final score is based on the number of apples eaten by the snake.

CONCEPTS APPLIED

- ➔ **ActionListener**
- ➔ **Swing**
- ➔ **AWT**
- ➔ **Graphics 2D**
- ➔ **Overriding**
- ➔ **KeyListener**
- ➔ **ActionEvent**
- ➔ **IOException**
- ➔ **PrintWriter**
- ➔ **FileReader**
- ➔ **FileWriter**
- ➔ **Scanner**
- ➔ **PrintWriter**

IMPLEMENTATION

First, we will define the constants used in our game.

```
private final int DOT_SIZE = 10;    // 300 * 300 = 90000 / 100 = 900

private final int ALL_DOTS = 900;

private final int RANDOM_POSITION = 29;
```

The DOT_SIZE is the size of the apple and the dot of the snake. The ALL_DOTS constant defines the maximum number of possible dots on the board (300 * 300 = 90000 / 100 = 900). The RANDOM_POSITION constant is used to calculate a random position for an apple. The DELAY constant determines the speed of the game.

```
private final int x[] = new int[ALL_DOTS];

private final int y[] = new int[ALL_DOTS];
```

These two arrays store the x and y coordinates of all joints of a snake.

```
private void loadImages() {

    ImageIcon iid = new ImageIcon("dot.png");
    ball = iid.getImage();

    ImageIcon iia = new ImageIcon("apple.png");
    apple = iia.getImage();

    ImageIcon iih = new ImageIcon("head.png");
    head = iih.getImage();
}
```

In the loadImages() method we get the images for the game. The ImageIcon class is used for displaying PNG images.

```
    public void initGame(){

        dots = 3;

        for(int z = 0 ; z < dots ; z++){
            x[z] = 50 - z * DOT_SIZE; // x[0] y[0] // x[1] y[1] // x[2] y[2]
            y[z] = 50;
        }

        locateApple();
```

```
        timer = new Timer(140, this);
        timer.start();    }
```

In the initGame() method we create the snake, randomly locate an apple on the board, and start the timer.

```
    public void checkApple(){
        if((x[0] == apple_x) && (y[0] == apple_y)){
            dots++;
           score++;
            locateApple();
        }
    }
```

If the apple collides with the head, we increase the number of joints of the snake. We call the locateApple() method which randomly positions a new apple object.

In the move() method we have the key algorithm of the game. To understand it, look at how the snake is moving. We control the head of the snake. We can change its direction with the cursor keys. The rest of the joints move one position up the chain. The second joint moves where the first was, the third joint where the second was etc.

```
for (int z = dots; z > 0; z--) {
    x[z] = x[(z - 1)];
    y[z] = y[(z - 1)];
}
```

This code moves the joints up the chain.

```
if (leftDirection) {

    x[0] -= DOT_SIZE;

}
```

This line moves the head to the left.

In the checkCollision() method, we determine if the snake has hit itself or one of the walls.

```
for (int z = dots; z > 0; z--) {


    if ((z > 4) && (x[0] == x[z]) && (y[0] == y[z])) {

        inGame = false;

    }

}
```

If the snake hits one of its joints with its head the game is over.

```
if (y[0] >= 900) {

    inGame = false;

}
```

This is the main class

```
setResizable(false);

pack();
```

The setResizable() method affects the insets of the JFrame container on some platforms. Therefore, it is important to call it before the pack() method. Otherwise, the collision of the snake's head with the right and bottom borders might not work correctly.

CODE

Board.java

```
package snake.game;


import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;
public class Board extends JPanel implements ActionListener{


    private Image apple;
    private Image dot;
    private Image head;
    private final int DOT_SIZE = 10;    // 300 * 300 = 90000 / 100 = 900
    private final int ALL_DOTS = 900;
    private final int RANDOM_POSITION = 29;
```

```
private int apple_x;

private int apple_y;

private final int x[] = new int[ALL_DOTS];

private final int y[] = new int[ALL_DOTS];

 private  int score = 0;

private int highScore = 0;

private boolean leftDirection = false;

private boolean rightDirection =  true;

private boolean upDirection =  false;

private boolean downDirection =  false;

private boolean inGame = true;


private int dots;


private Timer timer;


Board(){


    addKeyListener(new TAdapter());

    setBackground(Color.BLACK);

    setPreferredSize(new Dimension(300, 300));
```

```
    setFocusable(true);

    loadImages();
    initGame();
}

public void loadImages(){
    ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("apple.png"));
    apple  = i1.getImage();

    ImageIcon i2 = new ImageIcon(ClassLoader.getSystemResource("dot.png"));
    dot = i2.getImage();

    ImageIcon i3 = new ImageIcon(ClassLoader.getSystemResource("head.png"));
    head = i3.getImage();
}

public void initGame(){

    dots = 3;

    for(int z = 0 ; z < dots ; z++){
```

```
            x[z] = 50 - z * DOT_SIZE; // x[0] y[0] // x[1] y[1] // x[2] y[2]
            y[z] = 50;
        }

        locateApple();

        timer = new Timer(140, this);
        timer.start();
    }


    public void locateApple(){

        int r = (int)(Math.random() * RANDOM_POSITION); // 0 and 1 =>  0.6 * 20 = 12* 10 =
120
        apple_x = (r * DOT_SIZE);

        r = (int)(Math.random() * RANDOM_POSITION); // 0 and 1 =>  0.6 * 20 = 12* 10 = 120
        apple_y = (r * DOT_SIZE);
    }

    public void checkApple(){
        if((x[0] == apple_x) && (y[0] == apple_y)){
```

```
        dots++;
        locateApple();
    }
}

public void paintComponent(Graphics g){

    super.paintComponent(g);

    draw(g);
}

public void draw(Graphics g){
    if(inGame){
        g.drawImage(apple, apple_x, apple_y, this);

        for(int z = 0; z < dots ; z++){
            if(z == 0){
                g.drawImage(head, x[z], y[z], this);
            }else{
                g.drawImage(dot, x[z], y[z], this);
            }
```

```
            }


            Toolkit.getDefaultToolkit().sync();
        String msg3 = "SCORE IS " +score ;
        Font font = new Font("PLAIN", Font.BOLD, 16);
        FontMetrics metrices = getFontMetrics(font);

        g.setColor(Color.WHITE);
        g.setFont(font);
        g.drawString(msg3, (300 - metrices.stringWidth(msg3)) / 2 , 30/2);

        }else{
            gameOver(g);
        }
    }


public void gameOver(Graphics g){


        String msg1 = "Game Over ";
        String msg2 = "SCORE IS " +score ;
```

```
    Font font = new Font("SAN_SERIF", Font.BOLD, 14);
    FontMetrics metrices = getFontMetrics(font);

    g.setColor(Color.WHITE);
    g.setFont(font);
    g.drawString(msg1, (300 - metrices.stringWidth(msg1)) / 2 , 300/2);
    g.setColor(Color.RED);
    g.drawString(msg2, (300 - metrices.stringWidth(msg2)) / 2 , 250/2);
    g.setColor(Color.RED);

}
public void checkCollision(){

    for(int z = dots ; z > 0 ; z--){
        if((z > 4) && (x[0] == x[z]) && (y[0] == y[z])){
            inGame = false;
        }
    }

    if(y[0] >= 900){
```

```
            inGame = false;
        }

        if(x[0] >= 300){
            inGame = false;
        }

        if(x[0] < 0){
            inGame = false;
        }

        if(y[0] < 0 ){
            inGame = false;
        }

        if(!inGame){
            timer.stop();
        }
    }

    public void move(){
```

```
    for(int z = dots ; z > 0 ; z--){
        x[z] = x[z - 1];
        y[z] = y[z - 1];
    }

    if(leftDirection){
        x[0] = x[0] -  DOT_SIZE;
    }
    if(rightDirection){
        x[0] += DOT_SIZE;
    }
    if(upDirection){
        y[0] = y[0] -  DOT_SIZE;
    }
    if(downDirection){
        y[0] += DOT_SIZE;
    }
    // 240 + 10 = 250
}

public void actionPerformed(ActionEvent ae){
    if(inGame){
```

```
            checkApple();
            checkCollision();
            move();
        }

        repaint();
    }


    private class TAdapter extends KeyAdapter{

        @Override
        public void keyPressed(KeyEvent e){
            int key =  e.getKeyCode();

            if(key == KeyEvent.VK_LEFT && (!rightDirection)){
                leftDirection = true;
                upDirection = false;
                downDirection = false;
            }

            if(key == KeyEvent.VK_RIGHT && (!leftDirection)){
```

```
            rightDirection = true;
            upDirection = false;
            downDirection = false;
        }

        if(key == KeyEvent.VK_UP && (!downDirection)){
            leftDirection = false;
            upDirection = true;
            rightDirection = false;
        }

        if(key == KeyEvent.VK_DOWN && (!upDirection)){
            downDirection = true;
            rightDirection = false;
            leftDirection = false;
        }
    }
  }

}
```

Snake.java

```
package snake.game;

import java.awt.EventQueue;

import javax.swing.*;

public class Snake extends JFrame{

public Snake() {

        initUI();

    }

    private void initUI() {

        add(new Board());

        setResizable(false);

        pack();

        setTitle("Snake");

        setLocationRelativeTo(null);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            JFrame ex = new Snake();

            ex.setVisible(true);

        });    }}
```

SNAPSHOT

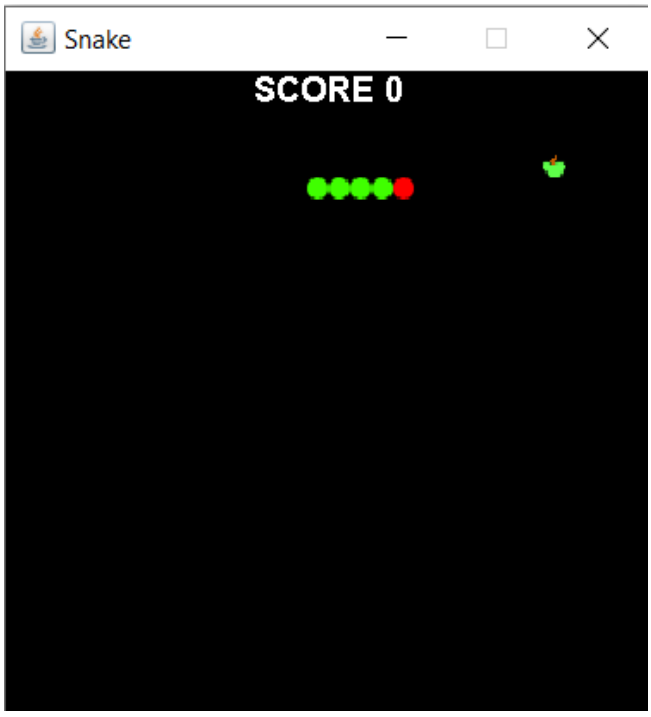

fig:1(start of the game)

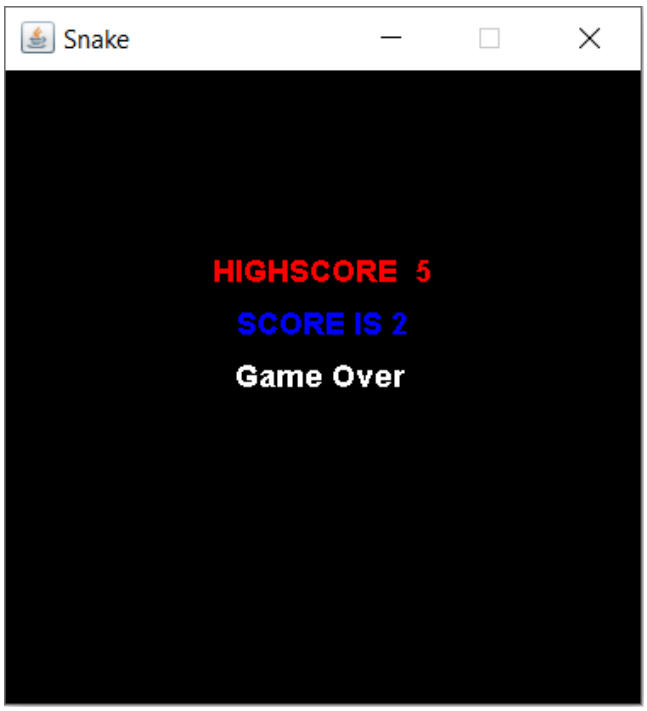

fig:2(after you lose)

References

Java: The Complete Reference by Herbert Schildt

Programming with Java A Primer by E.BalaGuruSwamy

Introduction to Java Programming by Y. Daniel Liang, Pearson, 11th edition, 2017

https://stackoverflow.com/

https://ssaurel.medium.com/learn-to-create-a-snake-game-in-java-c41e3e3b216e