JAVA PROJECT REPORT

(Project Term January-May 2023)

Platformer Game

Submitted by

Name : Rituraj Goswami **Registration Number : 12101798**

Course Code - CSE310

Under the Guidance of

Dr. Ranjith Kumar A

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project work entitled **Platformer Game** is an authentic record of my own work carried out as requirements of Capstone Project for the award of B.Tech degree in Computer Science Engineering from Lovely Professional University, Phagwara, under the guidance of Dr.Ranjith Kumar A , during January to May 2023. All the information furnished in this capstone project report is based on my own intensive work.

Name of Student : Rituraj Goswami

Registration Number: 12101798

Roll Number : A19

Section : K21WY

(Signature of Student)

Date:

TABLE OF CONTENTS

INTRODUCTION

Platformer games have long been a well-liked genre in the gaming industry. A player-controlled figure in these games often travels over a variety of levels, barriers, and opponents in order to accomplish a certain objective. Platformer games that are enjoyable and demanding may now be made thanks to the introduction of the Java programming language.

It is possible for game creators to construct intricate and engaging games using Java, a flexible and strong programming language. It is a well-liked option for creating video games due to its object-oriented architecture and cross-platform features. Games may be developed using Java for a range of platforms, such as consoles, mobile phones, and desktop computers

Java platformer games frequently include brilliant and colorful visuals, catchy music, and demanding gameplay. They provide an immersive experience that will keep gamers interested and delighted for hours. Platformer games created with Java provide a unique and interesting gaming experience, whether you're a seasoned gamer or a casual player.

LIST OF LIBRARIES USED

- Javax.Swing.JFrame
- Java.awt.color
- Java.awt.dimensions
- Java.awt.Graphics
- Java.awt.Graphics2D
- Java.awt.event.KeyEvent
- Java.awt.event.Keylister
- Java.awt.image.BufferedImage
- Java.io.IOExpectation
- Java.io.InputStream
- Java.io.InputStreamReader

MODULES USED IN THE PROJECT

(Their Explanation and Code)

1. Main.java

```java
package main;
import javax.swing.JFrame;
public class Main{
   public static void main(String[] args){
        JFrame window = new JFrame();
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false);
        window.setTitle("Xtainverse");

        GamePanel gamePanel = new GamePanel();
        window.add(gamePanel);

        window.pack();
        window.setLocationRelativeTo(null);
        window.setVisible(true);

      gamePanel.startGameThread();

   }
}
```

This Java code makes a straightforward game window utilizing the Java Swing library. The window is positioned to show a game board, which contains the really game rationale and illustrations.

The JFrame class is utilized to make a window, which is then modified utilizing different strategies. The setDefaultCloseOperation() strategy is utilized to determine what happens when the client taps the "X" button in the upper right corner of the window. For this situation, the window will close and the game will exit. The setResizable() strategy is utilized to keep the client from resizing the window, guaranteeing that the game looks and works the equivalent no matter what the window size.

The GamePanel class is a custom class that expands the JPanel class and contains the really game rationale and illustrations. The board is added to the window utilizing the add() strategy. The pack() strategy is utilized to change the size of the window to fit the items in the board.

The setLocationRelativeTo() strategy is utilized to focus the window on the screen, and the setVisible() technique is utilized to make the window apparent to the client.

At long last, the startGameThread() technique is approached the game board, which starts the game circle and starts the game running. This technique is apparently characterized inside the GamePanel class and contains the really game circle rationale.

2. GamePanel.java

```java
package main;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import entity.Player;
import tile.TileManager;
public class GamePanel extends JPanel implements Runnable{

    // screen settings
    final int originalTileSize = 16; //16x16 characters
    final int scale = 3;
    public final int tileSize = originalTileSize * scale; //48x48 tile
    public final int maxScreenCol = 16;
    public final int maxScreenRow = 12;
    public final int screenWidth = tileSize*maxScreenCol; //768 pixels
    public final int ScreenHeight = tileSize*maxScreenRow; //576 pixels

    //world Settings
    public final int maxWorldCol = 50;
    public final int maxWorldRow = 50;
    public final int worldWidth = tileSize * maxWorldCol;
    public final int worldHeight = tileSize * maxWorldRow;

    //FPS
    int FPS = 45;
    TileManager tileM = new TileManager(this);
    KeyHandler KeyH = new KeyHandler();
    Thread gameThread;
```

This Java code characterizes the GamePanel class, which expands the JPanel class and contains the really game rationale and designs for a basic 2D platformer game.

The class incorporates a few constants and factors that characterize the screen and world settings, as well as the game's FPS (outlines each second) rate. The TileManager class is likewise launched, which deals with the game's tiles and foundation. A KeyHandler class is utilized to deal with client input, and a Player class is characterized as the fundamental person of the game.

The GamePanel constructor sets the favored size, foundation tone, and center settings of the board, and adds a KeyHandler audience to the board to catch client input. The startGameThread() strategy makes another string for the game to run on.

The run() strategy contains the really game circle, which refreshes the game rationale and designs at a set FPS rate. The update() strategy is called to refresh the player's situation and other game data, while the repaint() technique is called to redraw the screen with the refreshed data. The Thread.sleep() technique is utilized to postpone the game circle to guarantee a predictable FPS rate.

The update() technique just calls the player's own update() strategy to refresh its position in light of client input.

The paintComponent() strategy is utilized to draw the game illustrations on the screen. The tileM.draw() strategy is called to draw the game's tiles, trailed by the player's own draw() technique to draw the player sprite. The Graphics2D object is then discarded.

Generally, this code makes a straightforward game board with a player character and tiles that can be explored utilizing client input, with the game running at a steady FPS rate.

3. KeyHolder.java

```
package main;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
public class KeyHandler implements KeyListener{
	public boolean upPressed, downPressed, leftPressed, rightPressed;
	@Override
	public void keyTyped(KeyEvent e) {}
	@Override
	public void keyPressed(KeyEvent e) {
		int code = e.getKeyCode();
		if(code == KeyEvent.VK_W) {
			upPressed = true;		}
		if(code == KeyEvent.VK_S) {
			downPressed = true;		}
		if(code == KeyEvent.VK_A) {
			leftPressed = true;		}
		if(code == KeyEvent.VK_D) {
			rightPressed = true;		}
	}
	@Override
	public void keyReleased(KeyEvent e) {
		int code = e.getKeyCode();
		if(code == KeyEvent.VK_W) {
			upPressed = false;
		}
		if(code == KeyEvent.VK_S) {
			downPressed = false;		}
		if(code == KeyEvent.VK_A) {
			leftPressed = false;		}
		if(code == KeyEvent.VK_D) {
			rightPressed = false;		}

	}
}
```

This Java code is a KeyHandler class that executes the KeyListener interface, which is utilized to deal with console occasions. At the point when a key is squeezed, the keyPressed() capability is conjured, which changes the related boolean worth to valid. At the point when a key is delivered, the keyReleased() capability is executed, which sets the important boolean worth to bogus.

The KeyHandler class is utilized in the GamePanel class to deal with the player character's development in this present circumstance. The boolean factors upPressed, downPressed, leftPressed, and rightPressed are utilized to recognize which bolt key is squeezed, and the player character is moved fittingly in the GamePanel class' update() capability.

4. Entity.java

```
package entity;

import java.awt.image.BufferedImage;

public class Entity {

	public int worldX,worldY;

	public int speed;

public BufferedImage up1, up2, down1, down2, left1, left2, right1, right2;

	public String direction;

	public int spritecounter = 0;

	public int spriteNum = 1;	}
```

This code provides a simple Entity class for creating game objects. It includes instance variables for the entity's position in the game environment as well as its movement speed. It also features instance variables for multiple pictures depicting the entity's look from various angles, as well as a counter for animating the entity's sprite.

This class is most likely meant to be expanded with more particular entity classes, such as a Player or Enemy class, which provide extra instance variables and methods for managing the creature's behavior.

5. Player.java

```
package entity;

import java.awt.Color;

import java.awt.Graphics2D;

import java.awt.image.BufferedImage;

import java.io.IOException;

import javax.imageio.ImageIO;

import main.GamePanel;

import main.KeyHandler;

public class Player extends Entity {


	GamePanel gp;

	KeyHandler KeyH;


	public final int screenX;

	public final int screenY;


	public Player(GamePanel gp, KeyHandler KeyH) {

		this.gp = gp;

		this.KeyH = KeyH;


		screenX = gp.screenWidth/2 - (gp.tileSize/2);

		screenY = gp.ScreenHeight/2 - (gp.tileSize/2);


		setDefaultValues();

		getPlayerImage();

	}

	public void setDefaultValues() {
```

```
            worldX = gp.tileSize*23;
            worldY = gp.tileSize*21;
            speed = 4;
            direction = "down";
      }

      public void getPlayerImage() {
            try {

                  up1 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_up_1.png"));
                  up2 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_up_2.png"));
                  down1 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_down_1.png"));
                  down2 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_down_2.png"));
                  left1 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_left_1.png"));
                  left2 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_left_2.png"));
                  right1 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_right_1.png"));
                  right2 =
ImageIO.read(getClass().getResourceAsStream("/player/boy_right_2.png"));



            }catch(IOException e){
```

The programme is divided into two classes: KeyHandler and Player.

The KeyHandler class implements the KeyListener interface and is in charge of detecting when the arrow keys are pressed and setting appropriate boolean flags.

The Entity class is tended by the Player, who represents the character on the screen. Its constructor accepts the GamePanel and KeyHandler objects to interact with the game window and keyboard input.
Instance variables of the Player class provide the character's position in the game environment, speed, and sprite pictures for each direction. Every game tick, the update() function is called, and it modifies the character's location based on the arrow keys pushed. Every game tick, the draw() function is invoked and draws the character's sprite picture onto the game window.

6. TileManager.java

```
package tile;

import java.awt.Graphics2D;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import javax.imageio.ImageIO;

import main.GamePanel;

public class TileManager {

	GamePanel gp;

	Tile[] tile;

	int mapTileNum[][];


	public TileManager(GamePanel gp) {

		this.gp = gp;

		tile = new Tile[10];

		mapTileNum = new int[gp.maxWorldCol][gp.maxWorldRow];

		getTileImage();

		loadMap("/maps/map02.txt");

	}

	public void getTileImage() {

		try {

			tile[0]=new Tile();

			tile[0].image =

ImageIO.read(getClass().getResourceAsStream("/tiles/grass.png"));


			tile[1]=new Tile();
```

```
                tile[1].image =
ImageIO.read(getClass().getResourceAsStream("/tiles/wall.png"));

                tile[2]=new Tile();
                tile[2].image =
ImageIO.read(getClass().getResourceAsStream("/tiles/water.png"));

                tile[3]=new Tile();
                tile[3].image =
ImageIO.read(getClass().getResourceAsStream("/tiles/sand.png"));

                tile[4]=new Tile();
                tile[4].image =
ImageIO.read(getClass().getResourceAsStream("/tiles/tree.png"));

                tile[5]=new Tile();
                tile[5].image =
ImageIO.read(getClass().getResourceAsStream("/tiles/earth.png"));



            }catch(IOException e) {
                e.printStackTrace();
            }

    }

    public void loadMap(String filePath) {
            try {
```

This is a Java code for a Tile Supervisor class that deals with the stacking and drawing of tiles in a game. The class contains a constructor that introduces the Tile exhibit and the mapTileNum cluster, and calls two different strategies: getTileImage() and loadMap().

getTileImage() is a strategy that peruses in tile pictures utilizing the ImageIO class and stores them in the Tile exhibit. Each tile has an extraordinary file number.

loadMap() is a strategy that peruses in a text record that contains a guide of file numbers relating to explicit tiles. The mapTileNum cluster is loaded up with these file numbers.

The draw() strategy draws the tiles onto the game board. It emphasizes through each file in the mapTileNum exhibit and computes the place of each tile on the screen utilizing the player's situation and screen size. On the off chance that a tile is inside the player's screen, it is drawn onto the game board utilizing the file number to recover the comparing tile picture from the Tile cluster.

Generally, this code considers the proficient administration and drawing of tiles in a game climate.