BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
1.	Madataly	Arish	2502049706

Course Code: COMP6510 **Course Name :** Object Oriented Programming

Class: L2AC **Name of Lecturer(s) :** Jude Joseph Lamug Martinez

Major: Computer Science

Title of Assignment: Coffee Shop Android App using Java API

Type of Assignment: Final Project

Due Date: 10th June 2022 **Submission Date:**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per the lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Arish Madataly

Table of Contents

“Run Run little monkey: Maze game using Java”

Name: Arish Madataly

ID: 2502049706

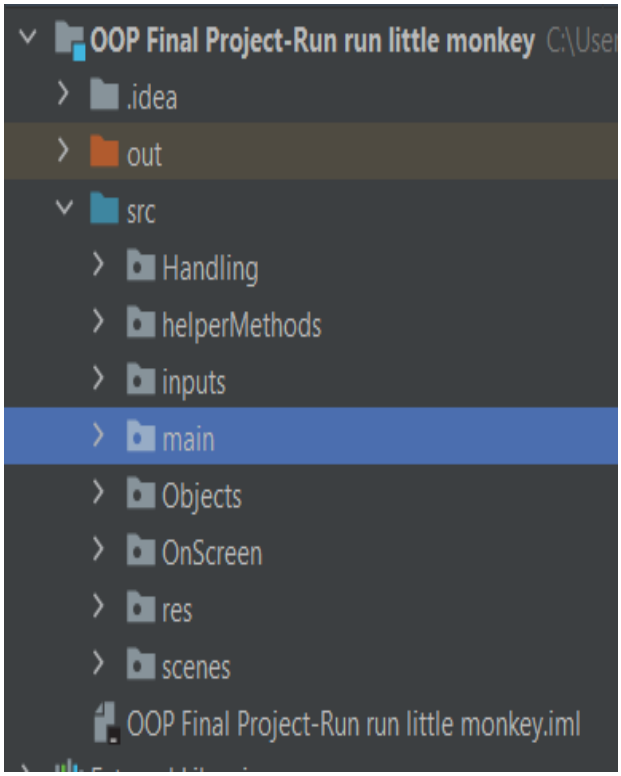
I. Program description

“Run Run little monkey” is a simple trap-based maze game where the player has to help a little monkey to find the right exiting tile in the shortest time possible. You can create any level that you would want by placing different tiles on the screen. There are road tiles where the monkey is allowed to walk, grass and decoration tiles where is not allowed to, and finally, tiles that have special effects when the player steps on them, these are called trap tiles. In the main game, the only visible tile will be the one where the player is standing, the rest will be hidden so the player will have to choose the direction he wants to go into by memorizing the places he has already gone to. This game was made fully using the java language.

This game is not like the other games where the player that hosts the game will be the one playing, it is more about creating different levels and letting your friend try and see how fast they can finish your maze. It’s fun and easy to play and even to edit a level, while the one creating the level can be very tricky and play around with the different special tiles. At the end, when the player that is trying to solve the maze finally succeeds he will see the full level as well.

II. General Organization of the code

The whole code is divided into packages that store different classes that are similar or have similar purposes:



main package: stores all the classes that set up the main mechanics of the game such as the game screen and the main thread

scenes package: stores all the scene classes that are on the different screens that have to be shown along with the game from the menu to the winning screen.

Object package: stores the Player class, the Tile class and the PathPoint class that are all objects used in the game. They are all part of the game and they are affected by or affect the player.

Handling package: stores classes that would handle an object, for now, the only object that needed to be handled was the Tile object because the game has a lot of different types of it and they all needed to be initiated. I still created the package to make the usage of this class clearer.

helperMethods package: stores final classes storing methods that are used for a few different utility purposes

in different parts of the code. Those classes will not have subclasses or be instantiated, they will just serve as utility classes. Those different methods consist of making changes in matching the type of data gotten to the type of data needed (converting functions). Writing into, creating or loading resources from outside the game such as images or text files. Storing constant variables that will later be compared to the actual values (this simplifies the use of switch statements). And finally, storing methods that are used to modify images.

Inputs package: stores classes that implement listener interfaces used to receive inputs from either the keyboard or the mouse.

OnScreen package: stores classes that are part of the UI, and everything visible on the screen such as buttons, and bottom bars.

res package: stores all the external files that are used in the game such as the images and the text files

audio package: stores a Sound class that deals with the audio.

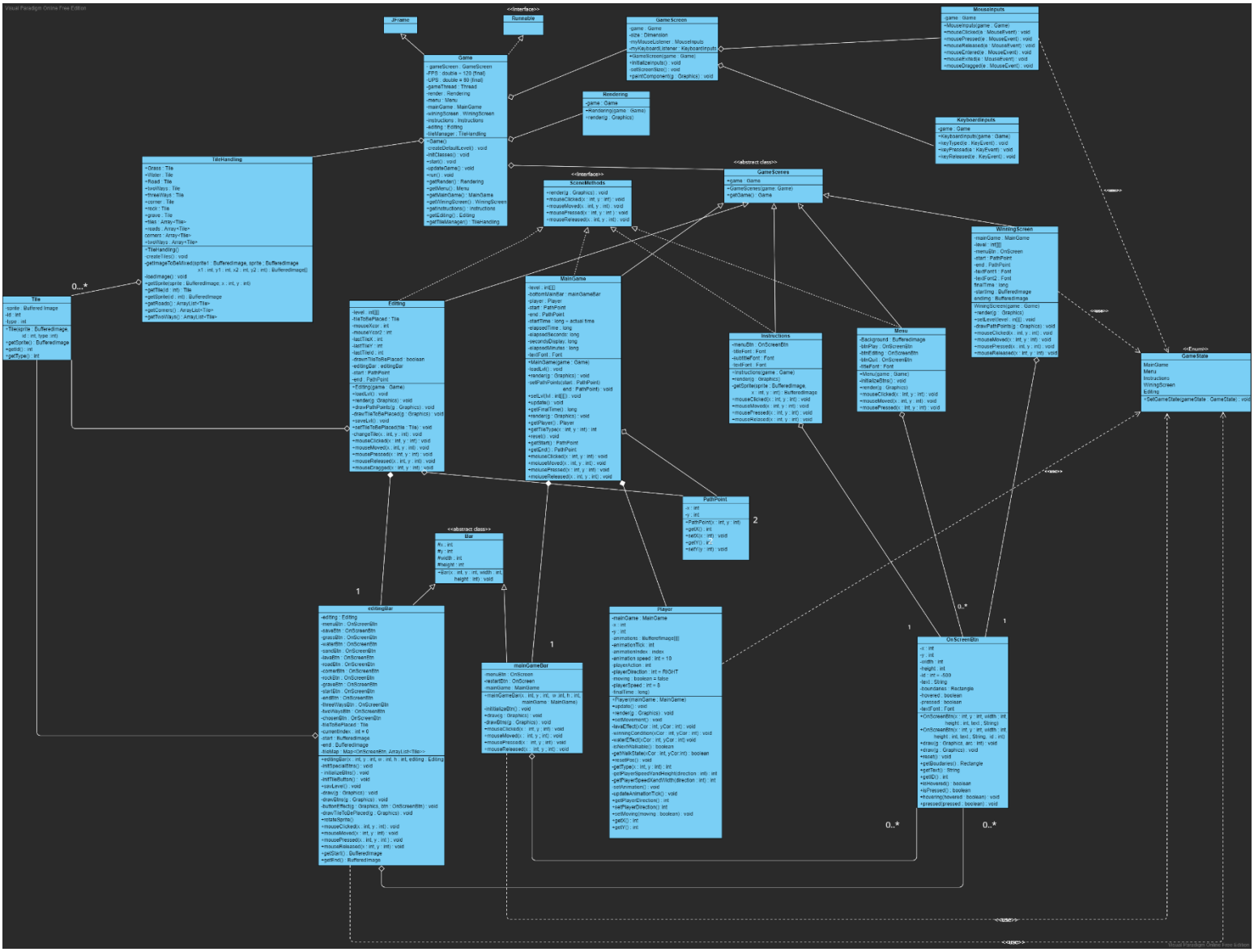
III. Class diagram

The code is made out of a lot of classes mainly because I did not want to put too many things in the main code so that it looks neater. But also, to make the code more organized as the classes with the same purpose are stored together. There are some classes that are actually very important to the game such as:

- Φ Game, GameScreen and Rendering which are the main classes in the code are the ones that take care of everything that happens on the screen such as the drawing and the updating part of the game.
- Φ Then there will be the scene classes that are all the different states of the game such as the instructions screen, the main game screen, the winning screen and the menu.
- Φ KeyboardInputs and MouseInputs to accept the inputs from the player and update the game accordingly.
- Φ Tile, Player and PathPoint are the objects present in the game.
- Φ The User Interface classes are all the things that the user will see on the screen and will be able to interact with, such as buttons and bottom bars.

And then few utility classes were also made in order to store the methods that would be used a lot in the code and also to make the code more organized. These classes are all final and they do not have instances, they are only used for the call of the methods that they contain.

- Φ The loadSave class gets resources from outside the game such as image and load them in. It also creates a file to store the level, and then load or save the changes into the actual game.
- Φ A constant class that itself stores static classes that will have stores constant values that will be used to check the object's states during the game
- Φ A ImageChange stores methods that will be called when an image needs to be modified in a certain way including rotation and combination of multiple images.
- Φ The UsefulMethods class with a name that is self-explanatory stores different kinds of methods that are needed throughout the game such as converting methods.



link: https://drive.google.com/file/d/1hY_lmowiDXGky6IMSGoG-uR3p0Pu7vTt/view?usp=sharing (

IV. Game mechanics

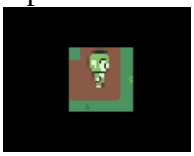
In this game, there are two types of users who have two distinct objectives. The first one would be the level maker; his work happens in the editing scene where he will place the different possible tiles wherever he wants on the 1000x640 editable screen. All the different types of tiles are shown on the bottom bar, clicking on them will let you move your mouse around the screen to decide where to place them.

Each type of tile has its own purpose and can be used to affect the player. The grass tile is the normal tile, the player cannot walk on it and it used to limit where the player can go on the map. The water tile will teleport the player to a random walkable tile on the map. The sand

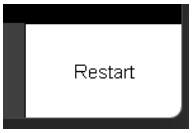
tile will slow the little monkey for as long as he is on that tile. The lava tile will instantly take him back to the starting point where everything restarts apart from the timer. The rock and the grave are just used as decoration they are not walkable and will most likely be seen only at the end of the level but it is always good to have a bit of aesthetics. The road tiles, this one is classified per type as well. This is the actual tile on which the player is supposed to walk, there are corners, intersections and straight roads. In addition to the ones that are shown on the screen, the road tiles can also be rotated using the right button of the mouse so that the possibilities are not limited. And finally, the path points are not really tiles as they will not be seen on the screen but they are listed there because the builder needs to specify where the player will appear and where the endpoint will be. A start and an end point are required in a level, they can only be placed on road tiles and they cannot be both located on the same spot. The level builder type of user's goal is to create the hardest level he possibly can by playing around with the different types of tiles, he needs to slow down and confuse the player as much as he can. To place a tile, you just need to decide on the position using your mouse and then click it, the new tile will automatically be placed at that point. It is possible to place multiple tiles at the same time as well, simply by dragging your mouse around the screen.

To finalize the level, the player can simply hit on save, the tile arrangement will be directly saved into a text file in the form of an array and will be loaded directly into the main game as well as on the winning screen.

The second type of user would be the maze solver, the one who will try to exit the maze by finding the end tile in the shortest time possible. This will happen in the mainGame scene where the little character will be drawn ready to move following the arrow key of the keyboard. In this part of the game, only a small portion of the map will be visible making it harder to find the end tile and it also stops the player from seeing the next tile, the only visible tile will be the one the character is standing on. The tile drawn on the screen will change as the player moves so that all the tiles around stay in the dark, it mimics a lamp effect where the whole maze is in the dark and you only have a lamp that can show you the tile you are standing on. The goal will be to control the monkey to go around the maze avoiding the special tiles and trying to find the exit while also minding the time. The different players can then compare their results.

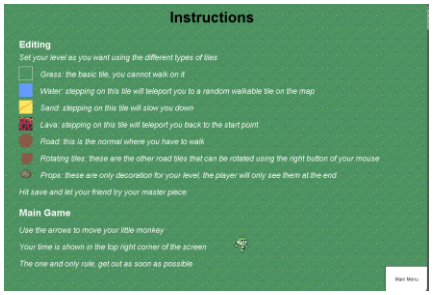


A restart button is also available on the bottom bar of the main game scene to allow the player to try from the beginning if he wants to. This button will reset the position of the player as well as the time, as this game is a time-based game there is no pause button available.

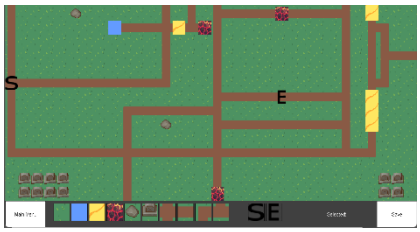


V. Game flow/Solution design

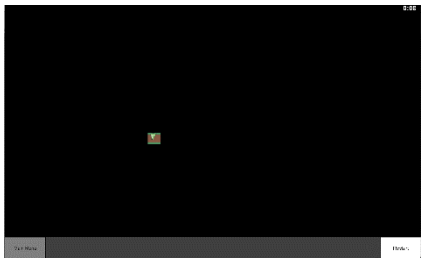
The game will open on the menu where the player can decide where he wants to go by clicking on the different buttons available. “Play” will direct him to the main game. “Edit” will direct him to the edit scene where he can build the level. “Instructions” will take him to a page explaining how the game works. While Quit will close the game.



The instruction page contains all the information about the editing part of the game as well as the actual playing part. All tile types are explained.



The edit scene is where the level is built before getting to the actual game as was stated before. After hitting save the level will be directly loaded into the game and ready to be played.



The level made in Edit can now be played in the Play scene where all the tiles are hidden. The user can use the arrow on the keyboard to move the little monkey.

When the player finally finds the exit tile, it will take him to the wining where a level reveal will occur with all the tiles being visible as well as his time on the bottom bar.

VI. Code description

Main classes

The game screen was built using the JFrame class which is a top-level container that provides a window on the screen where all other components will be added later. Almost every swing application starts with a Jframe window. The main class Game extends that top-level container and contains all the different scenes of the game as well as the GameScreen. It has all the getters necessary for the different instances of the different classes to interact with each other.

```
//Getters
public Rendering getRender() { return render; }

public Menu getMenu() { return menu; }

public MainGame getMainGame() { return mainGame; }

public WiningScreen getWiningScreen() { return winingScreen; }

public Instructions getInstructions() { return instructions; }

public Editing getEditing() { return editing; }

public TileHandling getTileManager() { return tileManager; }
```

This class also contain the main thread which is the path taken when the program is executed, it can also be referred to as the game loop. This thread is where all the updates and drawings happen through the game screen. It is also used to keep the frame rate and the update rate stable at 60 and 120 respectively.

The createDefaultLvl() method is a first-use method that creates a new text file and fills it with an array of 644 zeros that will represent the initial state of the level. This document will later be used to save the level's data and also to load them into the game.

The GameScreen which is a class that extends the JPanel, another built-in class that serves to attach other components to the window, is where all the drawing happens. This class represents the actual game screen drawn on the window made by the Game class. Usually, all the drawings and everything related to drawing on the screen is directly put into the painComponent() method of this class but to make the code more organized I made an extra class called Rendering that contains a render method. This method draws the different scenes based on a switch statement based on an Enum class called GameState that has as values all the game scenes. The GameState contains a setter method that allows the program to switch from one scene to the other. The Rendering class use the current value of the Game state to call the appropriate draw method from one of the functions, and then the GameScreen will call the render method to draw the method called on the screen and make it visible to the user.

```java
public void render(Graphics g) {
    switch(GameState.gameState) {
        case Menu:
            game.getMenu().render(g);
            break;
        case MainGame:
            game.getMainGame().render(g);
            break;
        case Editing:
            game.getEditing().render(g);
            break;
        case WiningScreen:
            game.getWiningScreen().render(g);
            break;
        case Instructions:
            game.getInstructions().render(g);
            break;
    }
```

```java
public enum GameState { // to set constants

    MainGame,
    Menu,
    Instructions,
    Editing,
    WiningScreen;

    public static GameState gameState = Menu;

    public static void SetGameState(GameState state) { gameState = state; }
}
```

This class also contains the instances of the input listeners which allow the screen to receive the user's input and react to these inputs by moving the different components.

Scenes

As referred to before, the scenes are what make up the whole game, the different menus are represented by these different scenes' classes. There are five different scenes the MainGame, the Editing menu, the Menu, the Instruction page and the Wining screen which all extend the abstract GameScene class and implement the SceneMethods interface. The super class is used to make all the scenes have an instance of the Game class so that they will be able to interact with the other objects and scenes through the getters. The interface is used to make sure all the scenes have the necessary methods which are the render() method that is required in order to draw the scene on the screen and all the mouse inputs methods that are necessary to allow the user's inputs.

Buttons

The buttons are an important UI component as they are used to travel from one screen to another, I decided to make my own button and set the effects myself. They are used all around the game with different functionalities. They are made using the class OnScreenBtn which takes as parameters the x and y for the position, the width, the height and finally the string for the text that will be written on it. It has two different constructors, one that takes an additional parameter which is the arc angle, this is for the buttons that are on the menu as they have rounded angles.

```java
if(hovered){
    g.setColor(Color.gray); //change color when hovered
} else {
    g.setColor(Color.yellow);
}
```

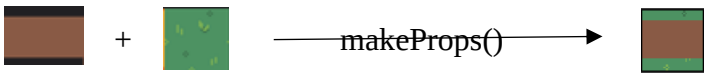
```java
if (pressed) {
    g.setColor(Color.black);
    g.drawRoundRect(x, y, width, height, arcWidth: 15, arcHeight: 15);
    g.drawRoundRect( x: x-1, y: y-1, width: width+2, height: height+2, arcWidth: 13, arcHeight: 13);
    g.drawRoundRect( x: x-2, y: y-2, width: width+4, height: height+4, arcWidth: 11, arcHeight: 11);
} else {
    g.setColor(Color.black);
    g.drawRoundRect(x, y, width, height, arcWidth: 15, arcHeight: 15);
}
```

Tiles

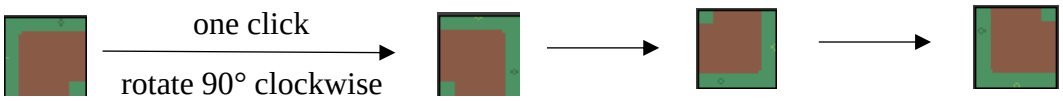
The tiles are the main object in this game as they constitute the level. There are different types and each of them will have a different effect on the player. The type is defined by an id that will be stored in a matrix that represents the level, and also by the type which is a constant integer that is unique for every type like the id. A tile is basically an image of 32x32 that represent a part of a level.

The TileHandling class was created to manage the tiles when they are used in the scenes, this class replace all the tile initialize that would need to be done in the actual scenes otherwise. It creates all the different tiles with the id and the type and the image.

There are some tiles that are made out of multiple images that are mixed using the makeProps() static method that takes an array of BufferedImage to then mixed them as they are added on top of each other on a 2D plan.



There are also tiles that can be rotated during the game, these are stored in an ArrayList that contains the same tile rotated over the 360°. The same image is rotated by multiples of 90° using the getRotateImage() which uses the Graphics2D rotate() to change the image. When the player wants to rotate an image he will just need to click on the right button of his mouse, this action will call the rotateSprite() method of the editingBar which will go through a map that takes the certain tile as a key and return an ArrayList as the value, each click will then be used to traverse the array of buffered images.




```
if (tileMap.containsKey(chosenBtn)) {
    currentIndex++;
    if (currentIndex >= tileMap.get(chosenBtn).size()) { //if t
        currentIndex = 0;
    }
    tileToBePlaced = tileMap.get(chosenBtn).get(currentIndex);
    editing.setTileToBePlaced(tileToBePlaced);
}
```

File

There are three different steps used to manage the file where the level matrix is saved. All the methods that affect the file are stored in the utility final class loadSave, when they need to be used the class is imported and called.

The first step is to create the level, if the machine hosting the game doesn't have a file in its resource folder yet, then the static method CreateLevel() is called to create a new file. Then the WtoFile() method which prints an array of values into the doc will be used to write an

array constituted of 644 zeros into the file. This will determine the default level by using each data as the id.

The loadLvl method which is present in the Editing, MainGame and Wining screen constitutes one of the main functionalities of Run run little monkey. It also constitutes the second step of file management. It gets the data from an external file by calling the getLevelData() from the utility class. This method is itself constituted by multiple other methods. The RFile() method will use the Scanner class to go through the level file and store the data in an ArrayList. The first 640 values in this file will be converted into a 2D array of length 20 representing the height of the playable screen and an inner array length of 32 representing its width. The conversion will be executed by the convertArrayListTo2Darray() method. This new 2D array will then be assigned to the level variable that will be used to draw the screen. The last 4 values of the file will be used as the path points coordinate as well retrieved using another method called getLevelPathPoints() which works the same way.

```
public static int[][] getLevelData(String name){
    File lvlFile = new File( pathname: "src\\res\\" + name + ".txt");

    if(lvlFile.exists()){
        ArrayList<Integer> dataList = RFile(lvlFile);
        return UsefulMethods.convertArrayListTo2Darray(dataList, ySize: 20, xSize: 32);
    } else {
        System.out.println("File " + name + " does not exist");
        return null;
    }
}
```

Images

The images used in the game are loaded from the res package using InputStream which is an abstract class used to get input from external sources. All the images are loaded into the loadSave class and then accessible by other classes through its methods that act as getters.

```
public static BufferedImage getSpriteGround() {
```

```
InputStream is1 = loadSave.class.getClassLoader().getResourceAsStream( name: "res/Map_tiles.png");
try {
    imageGround = ImageIO.read(is1); // try to import the image
} catch (IOException e) {
    e.printStackTrace(); // if there is an error it will be displayed
} finally {
    try {
        is1.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

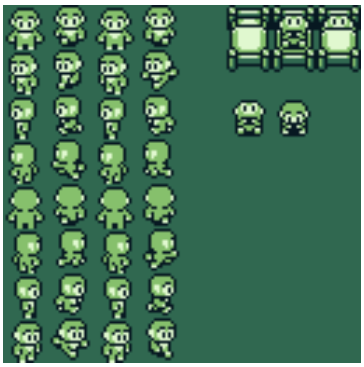
Audio

As this game is a maze resolution that requires attention, the only audio added was the background music and a winning ovation. These audios were added through the Sound class that uses Clip, a java built-in interface that loads the audio in the game it gets played. The URL built-in class loads a file path and the AudioStream that allows the audio file to be loaded into the game. The Sound has four main methods setFil() which defines the file that will be loaded, play() that start the audio in the game, the loop() that will play the audio until it's over and then restart it from the beginning, and finally the stop() that will stop the song that is actually playing.

Player

The player is used only in the main game scene represented by a little monkey that runs around the map trying to find the exit. The sprites are also loaded in the loadSave class, but they are retrieved into a 2-dimensional array that will take a constant named playerAction as the first index and the animationIndex variable. The playerAction changes depending on the direction the player is going into. This variable is defined in the PlayerConstants static class located in the constant class. The values of each possible value represent the location of the image in the sprite from top to bottom. While animationIndex is a value that increases by one each time the character move, it resets when it gets to 3 which is the total number of sprites per direction – 1.

```
public static final int RUNNINGDOWN = 0;
public static final int RUNNINGLEFT = 2;
public static final int RUNNINGUP = 4;
public static final int RUNNINGRIGHT = 6;
```



The character's possible movements are determined by constants that are assigned as the type of tile. The position of the player is determined and from there we can determine the corresponding tile in the level's matrix, by getting the type of that tile the possible movements of the player are assigned. To assign the possible directions, the method getWalkState() was used, it returns a boolean value that will either allow the player to go in a certain direction or not. For example, if the tile is a vertical road, the only possible directions are up and down, while when the road is an intersection he can go in any direction.

```
public static final int ROAD_VERTICAL = 6;
```

```
case ROAD_VERTICAL:
    return playerDirection == DOWN || playerDirection == UP;
```

```
public static final int ALL = 8;
```

```
case ALL:
    return playerDirection == UP || playerDirection == DOWN || playerDirection == LEFT || playerDirection == RIGHT;
```

When the first check returns true, there is now a second check to know if the next tile is one of the walkable tiles or not, this is done by adding a value to the x and y coordinates of the player. The value added to the x and y should be the same in every direction but due to a sprite offset, it had to be decided manually by drawing a square next to the player as seen in the image.

When both checks are positive that means the player can now move forward. And that check is done for each and every tile while the game is running. The check for the actual tile and the next one are the main mechanics around the player as the special tile's effects are also based on these methods.

```
private boolean isNextTileWalkable() {
    int nextX = x + getPlayerSpeedXandWidth(playerDirection);
    int nextY = y + getPlayerSpeedYandHeight(playerDirection);

    if (getType(nextX, nextY) > 2) { //if the tile is a special tile or a road
            //due to the sprite offset each direction need a value that will adjust the point that will define the position
            if (playerDirection == UP) {
                return getWalkState( xCor: x + 20,  yCor: y + 22);
            } else if (playerDirection == DOWN) {
                return getWalkState( xCor: x + 15,  yCor: y + 10);
            } else if (playerDirection == RIGHT) {
                return getWalkState(x,  yCor: y + 22);
            } else {
                return getWalkState( xCor: x + 20,  yCor: y + 22);
            }
        } else {
            return false;
        }
```

The character appears on point S which is determined when editing the level, it is the start of the maze and it can be placed only on road tiles. Similarly for the end tile, when the player gets to this tile that can be on any road on the map, he wins and the winning screen will appear.

Editing

The editing part of the game is actually the most important one because this is where the user hosting the game will spend the most time. As was stated before the whole level is loaded from the file into a two-dimensional array with each value representing the id of the tile situated at that y and x coordinate. The bottom bar will have all the different tiles that you can place and you can simply click on them. Then you can navigate on the screen with the tile you want to place, and with a left-click, the tile will be replaced, you can also drag which will do the same as well. As all the tiles are also buttons, they all have to be initialized that way.

VII. Possible improvements

I think the game mechanics are exactly what I wanted when I started so I am happy with the result. A few improvements that can be made I think are the sprites, they can be better

managed, more sound effects will enhance the player's experience and having more types of tiles because I think right now the game is a bit simple, it would be better to make it a bit complex. But overall, I think everything went well.

VIII. Possible Extensions

My first idea was to actually use a zoom-in function instead of making the rest of the map dark, so it would be fun to see the implementation of that as well. Also with a zoom, the player would be able to see the decoration around the map.

Another possible extension would be to hide the trap tiles and leave the whole maze visible, this would require memory work from the user that will have to remember where those traps are placed.

Adding more sound effects would also add more to the game, having sound effects for each type of effect.

And finally, I would like the extend the game by allowing the user to save multiple levels in different files and maybe allow him to send it to other people that can log the file on their computer. It would be amazing to actually implement the sharing function.

IX. Lesson learned

The process of making this game thought me a lot of things about the java language, how to use interfaces and abstract classes. It also showed the multiple differences between an object-oriented language like java and python, the fact that java is a compiled language makes it faster but it also needs more code to perform the same task as python would in way fewer lines. And finally, I learnt a few useful tips on how to locate bugs and correct them as I ran through a lot of them while making my game, since java requires making classes for almost everything it is a bit harder to debug because of the flow of the code.

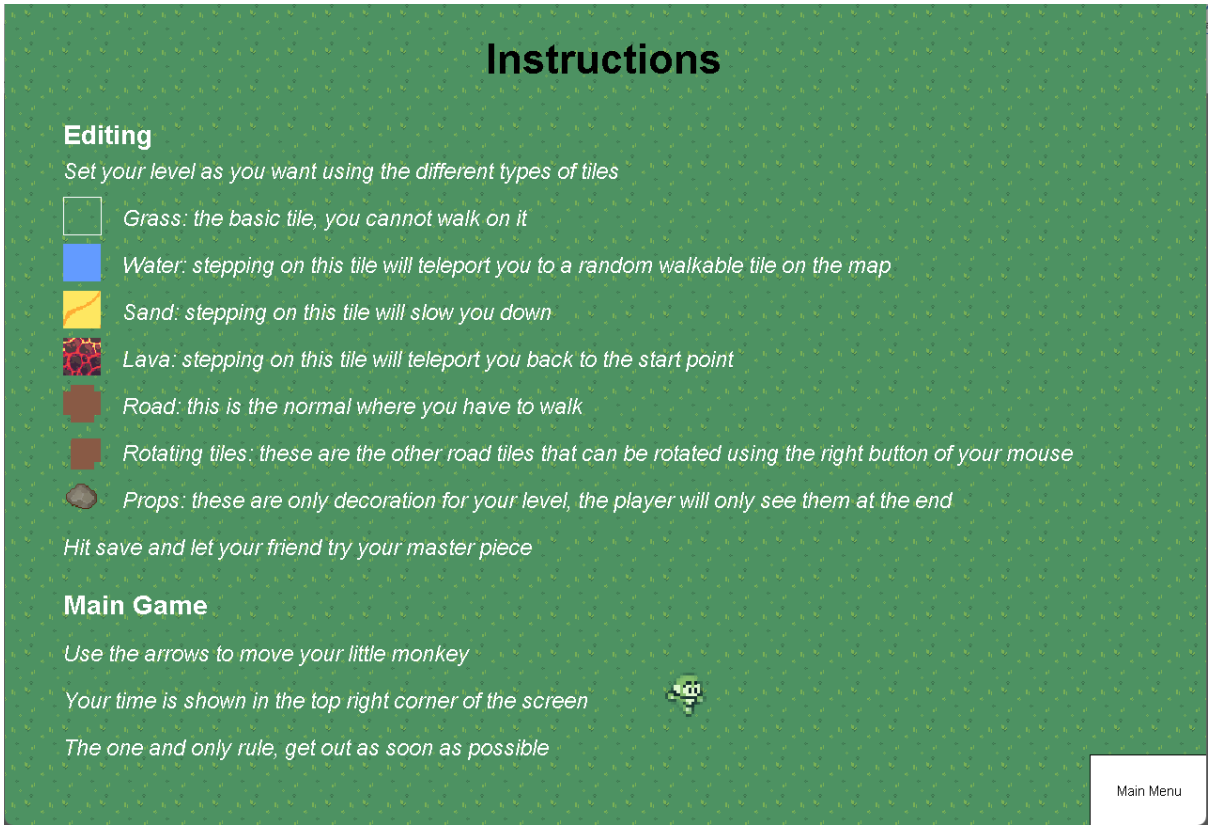
X. Proof of working program

Menu

Play

Edit

Instructions



Wining screen

References and resources

https://cloudconvert.com/mp3-to-wav

https://www.chosic.com/

https://mixkit.co/

https://itch.io/game-assets/free

https://craftpix.net/freebies/

https://stackoverflow.com/