Magasinière- A Puzzle in JAVA

OOP Semester Project

Tafheem Ul Islam Malik
CMS: 321906
BSCS-9, SEECS
NUST, Islamabad
tmalik.bscs19seecs@seecs.edu.pk

Abdul Manan
CMS: 303323
BSCS-9, SEECS
NUST, Islamabad
amanan.bscs19seecs@seecs.edu.pk

Hammad Saleem
CMS: 306647
BSCS-9, SEECS
NUST,Islamabad
hsaleem.bscs19seecs@seecs.edu.pk

***Abstract*—Magansiniere is a 2d puzzle game developed in Java. The game is executed in JavaFX Graphical User Interface. The coding part of the game contains several packages stacked with our classes as well as assets (media files etc.). There are four windows in the game- MainMenu (primary window which we run to get into the game), Instructions, LevelsMenu (to pick the level) and Main (back engine and the window where user plays the game). Several Object-Oriented concepts like inheritance, encapsulation, file handling and more are used in the project. This project was the best that would have made us understand and practice the concepts of Java.**

I. Introduction

A. Overview:

The report discusses the results of work done on the project, Magasiniere; a game developed as the semester project for class of Object-Oriented Programming (OOP) in Java. In this project students were expected to formulate and specify a small programming project, develop an appropriate piece of software to carry out the stated purpose by implementing the concepts of object-oriented programming that were taught in the course and provide a report on the outcomes of their project.
This report aims to provide a detailed look at the resulting application and analysis of the project design and development.

B. Document Conventions:

This document will interchange the pronoun 'we' with team's acronym as the project is result of efforts of all members of team. There is a clear distinction, however, between the use of words "player" and "user". The player is in-game character avatar being manipulated whereas the user is human being interacting with the game.

C. Background:

At the beginning of the project, we had no experience with GUI development in Java environment. So, a significant portion of time was dedicated to learning, investigating, understanding and testing smaller bits of functionality of JavaFx which is a library in Java used for building GUI application. Also, we spent a large portion of our time and efforts on designing and development of game.
Before jumping to any tool, we tested some alternatives for efficiency and productivity of project execution. Ultimately, we end up having IntelliJ Idea as IDE for our project development, JavaFx as our library for GUI and Adobe Photoshop and Canva for designing and editing some aspects of assets used in game.

D. Project Brief:

Magasiniere is a game developed in Java 8. The design of game is inspired from a classical Japanese game called Sokoban. Magasiniere falls under the genre of puzzle. The word is from French language which means 'Warehouse keeper'. The design of the game is simple and sophisticated, but the game is quite challenging to solve. In this game, a player pushes crates around a warehouse, trying to keep them in their positions.

E. Rules:

The game is played on a board of squares, where each square is a floor or a wall. Some floor squares contain crates and some floor squares are marked as storage locations. The player is confined to the board and may move horizontally or vertically onto empty squares (never through walls or crates). The player can move a box by walking up to it and push it to the square beyond. Crates cannot be pulled, and they cannot be pushed to squares with walls or other crates. The number of crates equals the number of storage locations. The puzzle is solved when all crates are placed at storage locations.

II. Method

A. Development:

1) Tools used in the development :

Following are the tools and libraries we used for developing our project.

a) IntelliJ Idea as IDE :

b) JavaFx for creating GUI :

Before settling on any tools, we tested different alternatives. We have tested our project on Eclipse too, though it worked but due to a simple interface and friendly environment of IntelliJ Idea, we finally used it as our IDE for the project. Similarly, we tested Swing, a GUI library in Java, before taking JavaFX as our primary library for the development of the game. We also tried using Scene builder, a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding. But then again, we realized that

our project can work fine without the use of this tool. So, to make our project simpler and constructive, we omitted the use of multiple tools and eventually settle down to use few but powerful tools.

2) Features :

The game incorporates the following features

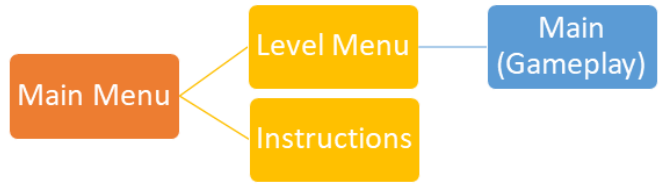
- Beautiful and simple GUI
- 15 Levels
- Level Menu to select any level
- Option to move to next and previous level during gameplay.
- Option to move to Main Menu in gameplay.
- Option to restart the game.
- Different styles of character when moving.
- Indication of placing crate on area.
- Background Music
- Choice to mute Background Music.
- Exit Button

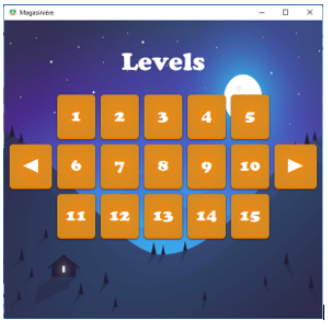
3) Game Overview:

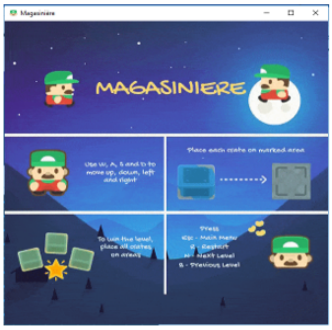
The project consists of four windows:

- Main Menu window
- Level Menu window
- Instructions Window
- Main (gameplay) window

The flow or hierarchy is as follows:







4) Game Details:

The game is designed as a board having 10 vertical and 10 horizontal blocks, thus creating a grid of 10x10. Each block displays an element of the game that can include walls, area (designated location to place crates on), Player, crate, floor tiles, and external ground tiles. In each level, there is single-player and multiple numbers of crates that are to be placed on the marked area. Crates can only be pushed to the blocks where there are no walls or other crates. The player is confined to move only on empty blocks such as over floor tiles. There are 15 levels of the game and the difficulty of levels increases with each completed level. Each level comes with different patterns of the setting of elements of the game. The basic goal of each level is to place all the crates on the marked area. To play the game use W,A,S,D keys to move the player, 'R' to restart the level, 'B' to go back to the previous level, 'N' to go to the next level and 'ESC' to go back to the main menu.

The board is developed using concepts of 2D arrays. The array consists of 10 rows and 10 columns, where each cell of array denotes a square of the board. It contains values from 0 to 6, each value corresponds to some element of the game. In this way, it becomes far easier to add levels as the only job that is to be done is to add an array with values from 0 to 6. The coordinates of the array and the coordinates of the block are the same as well. For example, if the cell with coordinate 2x2 (row 2 and column 2) has a value that corresponds to the crate, then in gameplay, the square with the same coordinate will hold crate as well. The game is based on this basic logic, this simplifies the development of the game.

For making the game beautiful and more interactive, we used images for assets instead of default shapes of JavaFx. We used the ImageView class of JavaFx library and added images of each asset and background as well. To further simplify the engine of the game, we declared the 2D array of ImageView class with 10 rows and 10 columns and linked it to our basic array that encompasses the values of elements of the game. What we have done is created a layer of images of all elements of the game that lays over the array of levels.

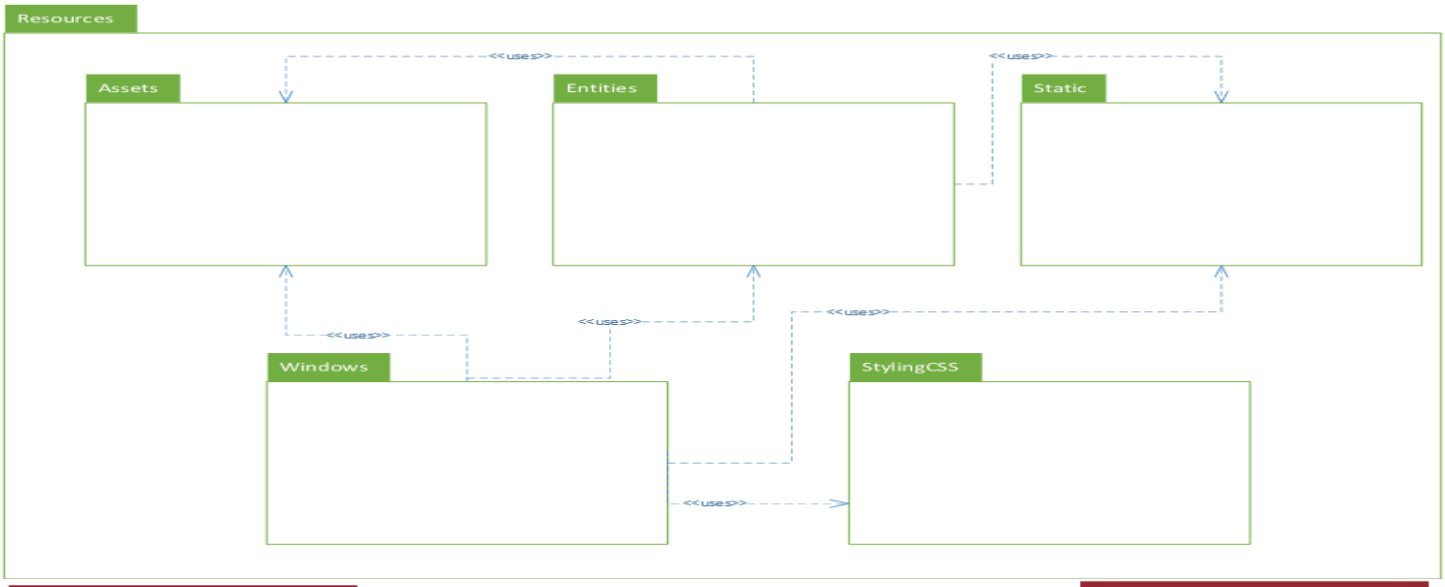
This is the basic logic on which the game's back engine is based on. This reduces the complexity of the development of the game. Further details of each class and working of the game are given in the following sections.

5) Description of Classes:

Before driving into analysis of classes of the project, to take look at the overview of all classes and their links between each other, the UML diagram of packages and classes is added below.

There are total five packages in the project all of which are stored in a single package Resources. The five packages are:

- Assets
- Entities
- Static



Data

+levelsPath : String
+sceneWidth : int
+sceneHeight : int
+blockWidth : int
+blockHeight : int
+title : String
+list : Arraylist<int>
+levelLoaded : Boolean
+levelsStoreTemp : int[][]
+levelsIntial : int[][][]
+addLevel (level: int[][]) : void
+getBlockSize() : void
+loadAllLevels() : void
+saveLevel() : void
println(s: String, x: int) : void

Main

primaryStageSaved : Stage
scene : Scene
root : AnchorPane
areaLayer : AnchorPane
cratesLayer : AnchorPane
playerLayer : AnchorPane
images : ImageView[][]
+crates : ArrayList<Crate>
areas : ArrayList<Area>
player : Player
+blocks : int[][]
+backAudio : Boolean
+level : int
levelMax : int
+rerun : Boolean
blockNames : String[]
blockHeight : double
+main() : void
+start(primaryStage : Stage) : void
+firstRun() : void
+restart() : void
createElements() : void
+getBackgroundImage() : void
+manageAllBlocks() : void
+getCrateAt(x: int, y: int) : Crate
setListeneres() : void
move() : void
getImages() : void
checkArea() : void
levelChange(i : int) : void
+checkAllCrateDirections() : void

LevelsMenu

primaryStageSaved : Stage
root : AnchorPane
scene : Scene
levelGrid : GridPane
levels : Button[][]
previous : Button
next : Button
label : Text
+backAudio : Boolean
+startingPoint : int
gridWidth : double
gridHeight : double
blockWidth : double
blockHeight : double
+start(primaryStage : Stage) : void
createElements() : void
resetButtons() : void
setListeneres() : void

LevelsMenu

primaryStageSaved : Stage
root : AnchorPane
scene : Scene
vbox : VBox
buttons : Button[]
label : Text
+start(primaryStage : Stage) : void
createElements() : void
setListeneres() : void

Sounds

+path : String
+sound : Media
+mediaPlayer : MediaPlayer
+mediaPlayer2 : MediaPlayer
+playsButtonAudio() : void
+playsBackgroundMusic() : void
+playsLevelAudio() : void
+playsBackgroundMusic() : void
+playsNextAudio() : void
+playsPreviousAudio() : void
+playsRestarttAudio() : void
+playsCrateAudio() : void

Player

+x : int
+y : int
+up: Boolean
+down : Boolean
+right : Boolean
+left : Boolean
+Player(x: int, y: int) << constructor >>
+setPosition(x: int, y: int) : void
+getPlayerImage(x: int) : void
+checkDirections() : void
+moveUp() : void
+moveDown() : void
+moveRight() : void
+moveLeft() : void

Crate

+x : int
+y : int
+up: Boolean
+down : Boolean
+right : Boolean
+left : Boolean
+Crate(x: int, y: int) << constructor >>
+getCrateImage() : void
+checkDirections() : void
+moveUp() : void
+moveDown() : void
+moveRight() : void
+moveLeft() : void

Instructions

+primaryStageSaved : Stage
+root : AnchorPane
+scene : Scene
+backAudio : Boolean
+start(primaryStage : Stage) : void
createElements() : void
setListeneres() : void

Area

+x : int
+y : int
+covered : Boolean
+Area (x: int, y: int) << constructor >>
+getAreaImage() : void
+open() : void
+close() : void

<<uses>>
<<depends>>

- StylingCSS
- Windows

a) Assets:

Package Assets includes all the image, audio and gif files used in the game. It includes complete sprite sheet of character player, images of elements i.e. wall, area, floor, ground tiles. The audio files of background music and other sound effects like that of clicking buttons, pressing keys, completing level and moving crate are also placed in this package. The package is used by classes in Windows and Entities package. To change graphics of the game, this is the package that should be used.

b) Entities:

Entities package comprises of three classes Area.java, Crate.java and Player.java. Each of these classes corresponds to the elements of the game. This package consists of the foundational classes that are used for creating objects by classes in Windows package. Explanation of working of each class of this package follows.

i) Area.java:

The Area class is used for creating 'Area' objects. Area is the element of the game where the crate is to be placed in order to clear the level. This class imports Image and ImageView class of JavaFx library and Data.java from Static package. The Area class extends the ImageView class i.e. it is inheriting all the attributes of ImageView class. The function of this class is to create an area object, define its coordinates x and y so that it can be placed over there and get the image of area for the object.

Member data fields:

public int x;
public int y;

These are x and y coordinate of area. They are used for setting X and Y property of ImageView class for the image of area.

public boolean covered;

True if the area is covered by the crate, false otherwise.

Member methods:

public void open()
public void close()

These methods maintain the value of boolean covered. If the crate is placed over the area, the boolean covered for the object area is declared true in close otherwise open declares it false.

ii) Crate.java:

The Crate class is used to create 'Crate' objects on the board. The data has been encapsulated as we can see every variable in this class is declared as private and public methods in the class are written to set and get the values of variables. Crate is the element of the game that is to be placed on the 'Area' to go through the level. This class imports Image and ImageView of the JavaFX library, Data and Sounds of the Static Package, Main of the Windows package. This class inherits the ImageView class. The main use of this class is to create a crate define its co-ordinates and show a crate on these co-ordinates in the game window.

Member data fields:

public int x;
public int y;

These are the x and y coordinate of crates. They are used for setting X and Y property of ImageView class for the image of crates.

private boolean up=false;
private boolean down = false;
private boolean left = false;
private boolean right = false;

These check if the crate can be moved in the pressed direction.

Member methods:

public void getCrateImage();

This method gets the crate image from resources and sets it in place of crates.

public void checkDirections ();

This function turns the above boolean variables true if conditions of movability are met.

public void moveUp ();
public void moveDown ();
public void moveLeft ();
public void moveRight ();

These methods will move the crate to the respective direction. They do it by removing the crate from the current block and taking it to the block where the player wants the crate to be moved.

iii) Player.java:

This class is responsible for creating the player/character object that plays the game. Player class imports Image and ImageView of the JavaFX library, Data class of the Static and Main class of the Windows package. Player class extends ImageView class, thus inheriting all the attributes of ImageView class. This inheritance is due to the reason that Player class deals with images of player and it must set some properties of image like position of image on screen, its width and height etc. which is possible by inheriting those properties from ImageView class. The data class is imported because its properties like sceneheight and sceneWidth are used in this class. Also the data has been encapsulated as we can see every variable in this class is declared as private and public methods in the class are written to set and get the values of variables.

Member data fields:

public int x;
public int y;

These are the x and y coordinate of player/character. They are used for setting X and Y property of ImageView class for the image of player.

private boolean up=false;
private boolean down = false;
private boolean left = false;
private boolean right = false;

These four Booleans are used to check whether the player can move in a particular direction. The player is confined to move over floor tiles only. If there is a crate or wall in any direction, the value of boolean in that direction will be declared false by the program.

Member methods:

public void setPosition();

This sets the position of the player at co-ordinates x and y.

getPlayerImage(int i)

This method loads the image of the player. There are four different positions of player i.e. facing front, back, left and right. Here the parameter 'i' comes from the methods from Main class. This method loads the right image according to the movement of player. This is determined by value of i that is sent as argument.

public void moveUp ();
public void moveDown ();
public void moveLeft ();
public void moveRight ();

These methods will move the player to the respective direction. They do it by removing the player from the current block and taking it to the block where the player is moved by the user.

c) Static:

This package has two classes Data.java and Sounds.java. Both these classes consist of static data fields and methods. The classes have static data fields and methods because they are to be used by other packages without creation of objects. This package provides values such as size of windows, levels and audio files for other packages. Both classes of Static package are explained below.

i) Data.java:

Data.java holds and manages the data for all the packages. This class has data regarding size of blocks and scene and title of project. Data class deals with values and loading of levels too. The purpose of creating a separate class for holding values was to simplify the development of project. To change size of windows and levels instead of changing size of each window, simply move to Data class and change the values there.

Member data fields:

public static int sceneWidth = 600;
public static int sceneHeight = 600;
public static int blockWidth = 0;
public static int blockHeight = 0;

These variables hold the size of scene and blocks. The width and height of scene are declared 600 as default.

public static final List<int[][]> levels=new ArrayList< >();

This array list holds all levels in an ordered way.

public static int[][] levelStoreTemp = new int[10][10];

levelStoreTemp is used to create temporary level for the process of saving it in form of txt file in the project.

public static int[][][] levelsInitial = new int[][][];

It consists of 15 levels where each level has 10 rows and 10 columns. These rows and columns hold values to corresponds to some element of the game. The value and the element that they corresponds are given in the following table.

Value	Element
0	Floor tiles
1	External tiles
2	Wall
3	Area
4	Crate
5	Player
6	Crate and Area both at same block

Member methods:

public static void addLevel(int[][] level);

This method add array of level in the list of levels i.e. it stores each level array in a list of levels.

public static void getBlockSize();

Calculates block's width and height by dividing scene width and height by 10. In default case, both dimensions are 60 pixels.

public static void loadAllLevels();
public static void saveLevel();
static void println(String s,int x);

These methods use file handling. If the levels have not been stored in Levels.txt yet loadAllLevels calls saveLevel method and the saveLevel calls printLn method which writes a file and stores the values of level. These values are stored in a text file Levels.txt. If the level values are already stored in Level.txt, it reads them by using file input stream and adds them to the levels list.

In our program, loading level was possible from level arrays only. But we decided to use file handling for two main purposes. Firstly, we wanted to learn file handling and no option is better than doing practice ourselves in the project. Secondly, we were planning to introduce another feature in which the user could add the level in the game. We could not however build this feature due to lack of time and problems faced by one of our members in this difficult times of pandemic.

ii) Sounds.java:

Sounds class loads all the audio files of project and plays them. It imports Media and MediaPlayer classes of JavaFx. Its data fields and methods are static as they are used by other classes. It has static MediaPlayer objects that is used in multiple static methods in the class. Each method plays some audio. The audio played by each method is written in the name of method. For example, method playsBackgroundMusic plays the background music.

d) StylingCSS:

This package consists of single CSS file main.css which is used for styling different aspects of Main Menu, Instructions and Levels Menu windows.

e) Windows:

This is the most important package of the project. It has files of each window. Most of GUI work is done in this package. Each class of this package is interlinked with each other, so they are placed in a single package. Following classes are part of this package.

- Main.java
- MainMenu.java
- LevelsMenu.java
- Instructions.java

i) Main.java:

This class is the back engine of the game. This class uses Image and ImageView classes for images. It also uses Player.java, Crate.java and Area.java to create instances of each of these class. Main class also uses Data.java to get levels of the game and size and title of the scenes.

Member Data Fields:

AnchorPane root, areaLayer, cratesLayer, playerLayer;

These panes are created as layers on screen, root being the basic layer and area, crates and player layers for the elements of the game.

ImageView[][] images;

This holds images of assets that are used in game.

public static List<Crate> crates;
List<Area> areas;

These lists of objects holds respective objects created in main class.

Player player;

This creates an instance of player class.

public static int[][] blocks;

Blocks is a 2d array which copies the level array in itself that is being played.

Member methods:

public void firstRun();

This method only runs once in the beginning of the program. The job of this method is to initialize different panes, get size of blocks from Data.java, declare lists of crates and area, create an instance of player and loads the images array with null.png.

public void restart() ;

The restart method is called in start after firstRun method. In this method, the level is loaded into blocks array and different methods are called.

void createElements();
public void getBackgroundImage() ;
public void manageAllBlocks() ;
void getImages() ;

These methods are responsible for loading images of different elements in the image array. Their job is to get images and create elements on the screen.

public static Crate getCrateAt(int x, int y);
public static void checkAllCrateDirections() ;
void move(int i);
void checkArea();

These methods deal with functionality of the main class. They update the position of crate, check if the crate is moveable by using methods from crate class, move player and check whether the area is covered by crate.

ii) MainMenu.java:

The MainMenu class/window is our primary class which we run to get the program up and runnning. The MainMenu class also inherits the Application class as it is a JavaFX application. The MainMenu class imports some JavaFX classes and some project package classes.

In this class we've used an icon for our application. As MainMenu will open first when we run the program therefore we start the background music from here onwards. We've added 3 buttons namely New Game- will take the to LevelsMenu, Instructions-which will bring up the Instructions window and Exit- which will close the game. We've also added a check box that will allow the user to play or stop the background music. We've also styled the MainMenu by using main.css from the Styling package. We've set listeners for the buttons and checkbox to work.

iii) LevelsMenu.java:
The LevelsMenu class/window brings up the levels we have stored in our program. This class also inherits Application class as it is a JavaFX application/window. The LevelsMenu class imports classes from JavaFX and some project project package classes.

We have created an AnchorPane, a GridPane, 15 level buttons per page, 2 buttons 'prev' and 'next' to change the pages and a Text label. We have also created boolean backAudio because when we press 'ESC' in the LevelsMenu the background music should not start again and overlap with the already running one.

Member methods:

createElements();
resetButtons();

When the program is run createElements() will be called which in itself calls resetButtons() method. These set up the grid and add the required buttons to the grid. When any button is pressed setListners() will listen to the call and perform the required function.

iv) Instructions.java:
This window basically contains instructions for user for how to play the game. The Instructions class inherits the Application class as it is a JavaFX window. We have made a custom image on which instructions are written and the image is then imported to the class by using Image class. And then we added it to anchor pane. From the instructions menu the user can go back to the main menu by pressing the 'ESC' key. We have setup listeners for that to work.

B. Design:

1) Inspiration:
The design of the game is inspired from retro styled classic Japanese games. However, we have given modern touch to design of the game wherever possible. The character is inspired from a famous character in gaming 'Mario'. Maze or setting of the board of the game is inspired from that of warehouse. In fact, whole design is somewhat like that of a warehouse.

2) Tools:
We have used graphics and sounds in the game from external sources in a legitimate way. The credits to the artists in given in the following sessions. However, to fulfil our requirements, we edited some of the assets. The tools that we used for editing and creating graphics are these:

- Adobe Photoshop
- Canva

3) Levels:
Since this is a puzzle game, so it is hard to come up with levels design that are challenging and able to solve. So instead of designing our own levels we used the royalty free levels designed by François Marques [1].

4) Graphics:
We looked and tried multiple pack of assets for our game. The asset used in this game is created by Kenny Vleugels (Kenney.nl) [2]. This is the link to the license of usage of assets [3]. The gif in the background is taken from Google. It is created by Julia [4].

5) Sounds:
The sounds used in this project are all no copyright sounds. The button audios are downloaded from fesliyanstudios [5]. The background music is called Sneaky Snitch and is composed by Kevin MacLeod [6].

III. DICUSSION

The program basically was an implementation of the classic Sokoban game. When we started the project, we thought it would be easy to do but what we didn't know that there are a lot of things happening while solving the puzzle, switching the windows etc. And the pandemic made it worse because the project was a group project and we all were from different places hence it made communicating for this coding-based project difficult. But we kept going and finalized our game.
There were some additional things that we wanted to do but couldn't due to the pandemic. The features we were unable to include this time, we will complete them in near future. One of the features we will include is that we would give the user the authority to add new levels himself while in the game and save them for future. Another thing we will do is to save the best players (who reached highest levels) and store them. In our game all the levels are unlocked but what we will do in the future is that every player will start from level 1 and other levels looked until he completes the previous level.
Furthermore, we did a lot of research about how to make our game better and better. For example, initially we used one single image to represent the movements of the player but what we later did was we used different images for different directions. Right image when the player faces right and so on. In the coding part the doubts about lists, abstraction, inheritance, polymorphism, static and non-static, public and private, CSS were cleared.

IV. RESULTS AND CONCLUSION

The whole theme of the project/game revolved around using Object Oriented Programming in Java with the help of JavaFX Graphical User Interface (GUI). The project made us understand the concepts of static and non-static, private and public, encapsulation, inheritance, exception handling, Lists and how to implement all these things in a JavaFX GUI. We saw how to create multiple windows, use media files (audio, images, gifs), use keyboard to handle objects in the GUI. In addition to that we used fill handling to store the levels.
Our game consists of 4 windows namely MainMenu - where our program begins, LevelsMenu- where we pick a level to play, Main- where we play the game and Instructions- for the user to learn how to play the game.
In the MainMenu, the user gets to chose 3 options- New Game – will take the user to LevelsMenu window, Instructions – will

take the user to Instructions window and Exit – will end the game. In the LevelsMenu, there are currently 15 levels and the user must pick one to play. In the Main user will play the game. The user must move the player in a way that all boxes should be on the marked areas to complete the level. One mistake means you will have to restart the level to retry. We have used some background music which will play throughout the game. We have also included button press sounds as well as level completion sound.

REFERENCES

[1] Francois Marques, Sokoban Levels, http://sokoban.online.fr/
[2] Kenney.Nl. https://www.kenney.nl/
[3] https://creativecommons.org/publicdomain/zero/1.0/
[4] Julia Comet, Dribbble. https://dribbble.com/shots/3175945–omet
[5] Free Mouse Click Sound Effects — MP3 Download — FStudios https://www.fesliyanstudios.com/royalty-free-sound-effects-download/mouse-click-2
[6] Royalty Free Music https://incompetech.com/music/royalty-free/index.html?isrc=USUAN1100772.