INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

School of Computer Science and Engineering

-----*-----**

PROJECT REPORT

CATCH ME IF YOU CAN

ALGORITHMS AND DATA STRUCTURES (IT013IU)

Course by Dr. Tran Thanh Tung

TEAM MEMBERS	STUDENT'S ID
Tran Nguyen Phuc	ITCSIU21097
Nguyen Mach Khang Huy	ITCSIU21072
Nguyen Tuan Khoa	ITCSIU21140
Vo Tran Khanh Quynh	ITITIU21024

TABLE OF CONTENTS

LIST OF FIGURES

CONTRIBUTION TABLE

No.	Full Name	Student's ID	Contribution
1	Tran Nguyen Phuc	ITCSIU21097	25%
2	Nguyen Mach Khang Huy	ITCSIU21072	25%
3	Nguyen Tuan Khoa	ITCSIU21140	25%
4	Vo Tran Khanh Quynh	ITITIU21024	25%

ABSTRACT

Based on the classic tag game that many people played as children, Catch Me If You Can is a tag game created in Java. The team's purpose is to produce an entertaining game that uses the knowledge learned in the course Algorithms and Data Structures.

In this game, the player’s goal is to run away from the pursuer. If the pursuer approaches the player, they can use teleport to move to another location on the map. The user interface is created with familiar features that are user-friendly and can be easily improved in the future.

The game will be thoroughly described in this report. In the first section of the report, the goals and resources used for this project will be described. In the second part, the rules and design will be fully explained. The demo and results will be in the next section. Finally, the conclusion and upcoming works will be covered.

CHAPTER 1. INTRODUCTION

1. Objectives

The project's objective is to develop a playable game that incorporates the knowledge acquired in the Algorithms and Data Structures course, with a user-friendly user interface and a structure that can be easily modified and improved in the future by the members.

In short, this project aims to:

- Create a game for entertainment.
- Apply algorithms and data structures knowledge.
- Gain experience in game development, code optimization, and project management.

2. The tools used

- IDEs for programming and debugging: JetBrains IntelliJ IDEA.
- Code version and project management: GitHub.
- Communication & Weekly Meetings: Discord.

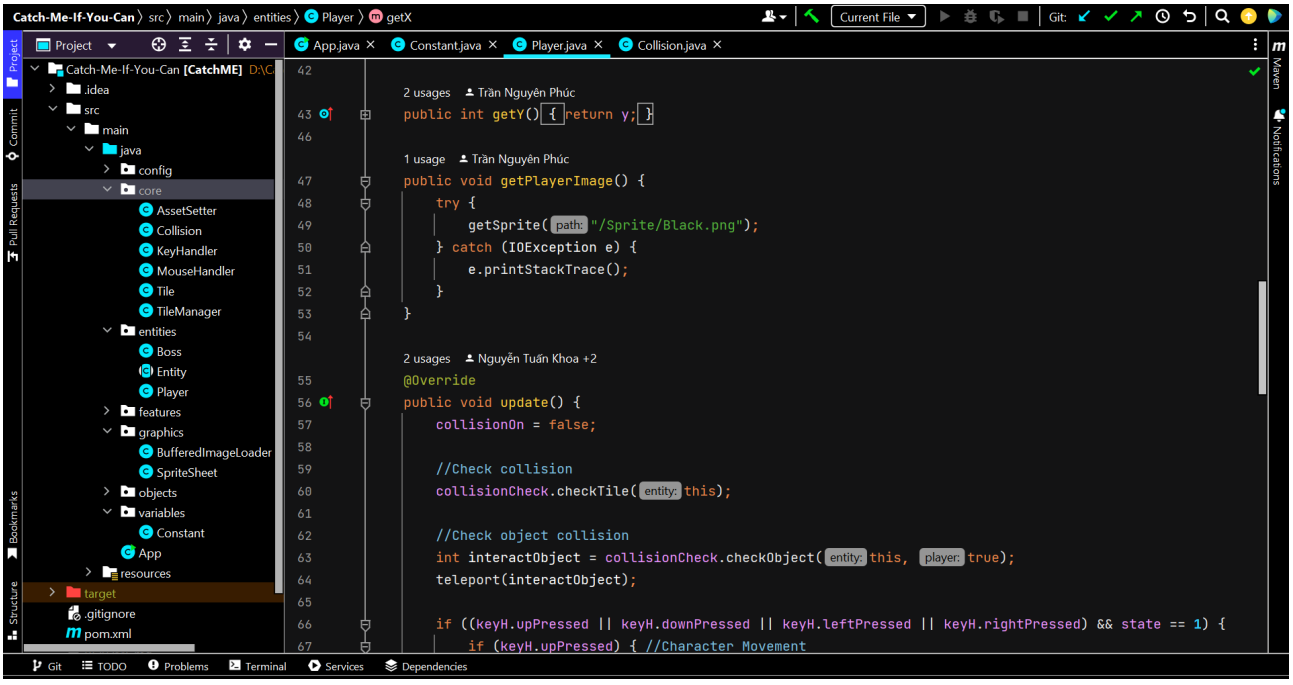

Figure 1.1. A class in JetBrains IntelliJ IDEA

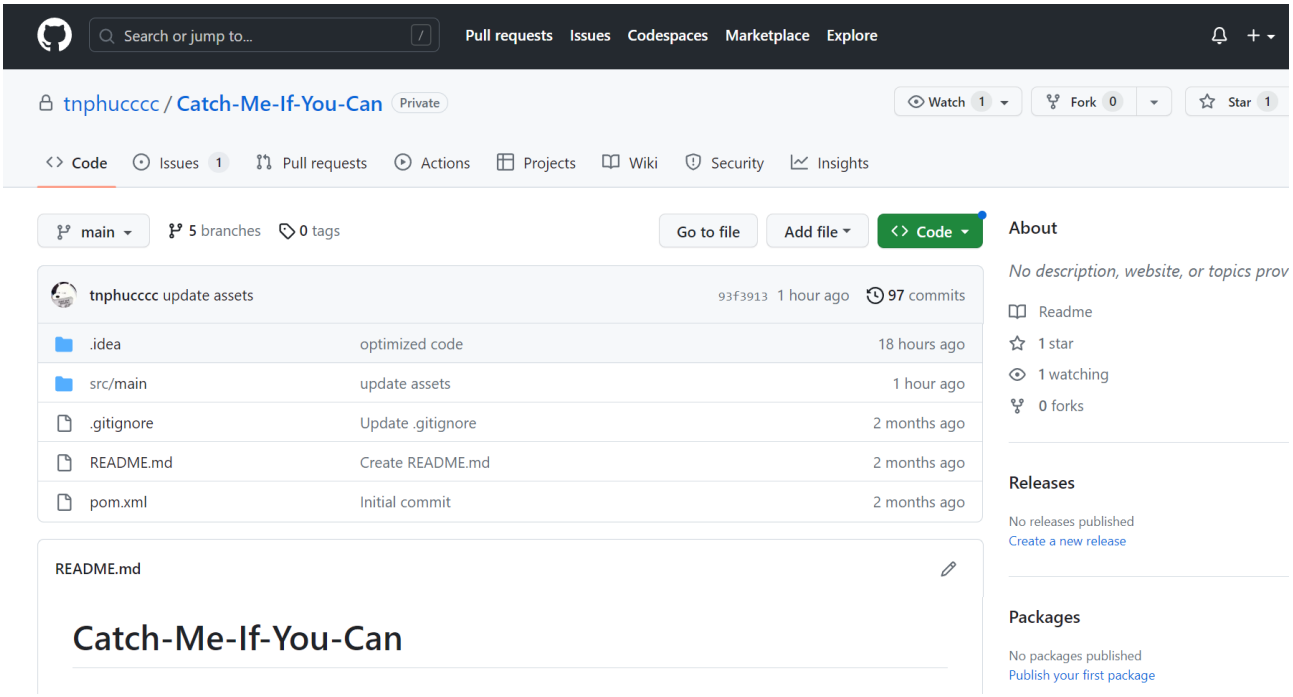

Figure 1.2. GitHub working environment

CHAPTER 2. METHODOLOGY

1. Rules

The team has decided to set the rules concisely and easily to understand for the player. This includes 3 main points:

- On the map, there will be a player and a boss who stand at a position far enough from the player. There are 2 types of teleport on the map: teleport for the player to go inside (teleport_in) and teleport where the player is released (teleport_out).

- The player's goal is to run away from the pursuer. If the pursuer approaches the player, they can use teleport to move to another location on the map. However, each teleport_in has its cooldown time which means that if the player goes inside one teleport_in, they will have to wait for a constant time to go into this teleport again.

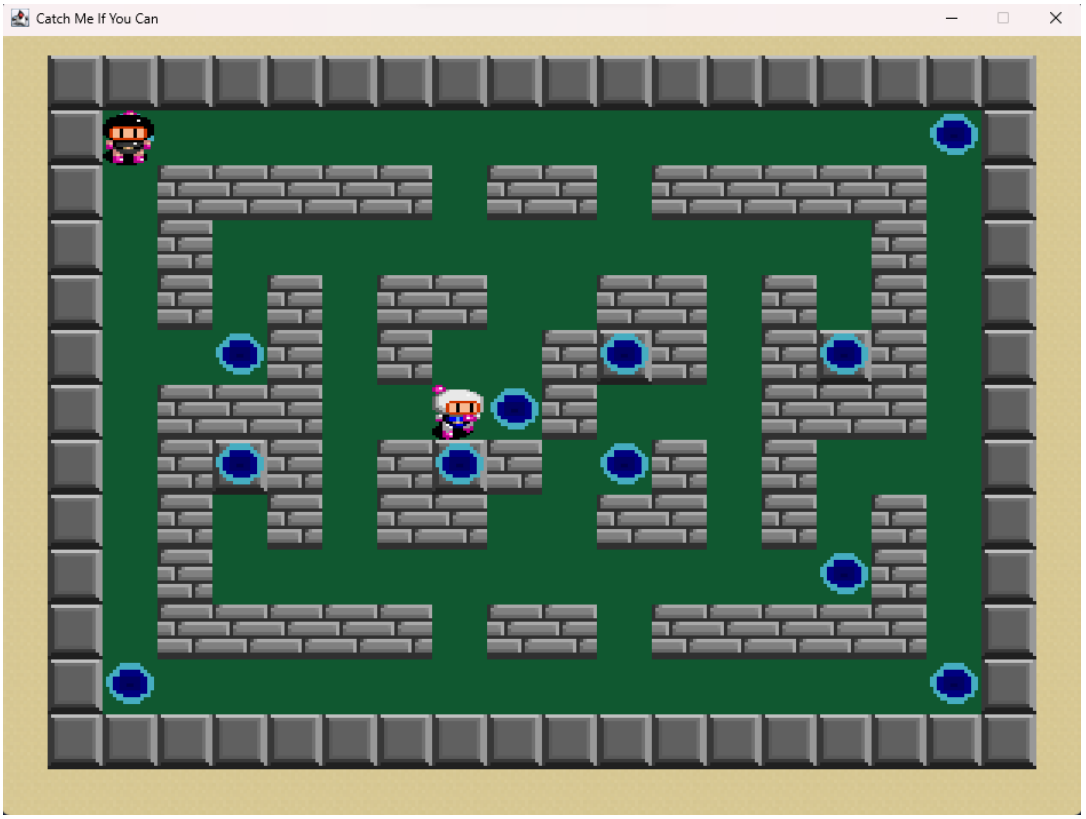

Figure 2.1. Original positions of player and boss

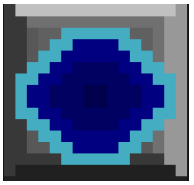

Figure 2.2. Teleport In

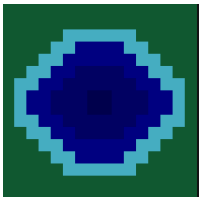

Figure 2.3. Teleport Out

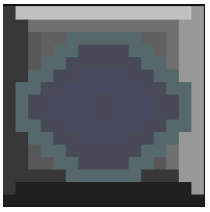

Figure 2.4. Teleport In (being cooldown)

2. Design

2.1. UI/UX

Based on our previous project of Bomberman Adventure, almost all the resources are identical, however, there still some changes have been applied to create a different style.

Our main character has been changed from black to white as in Figure 2.5. Meanwhile, the old sprite for the player became the sprite for the new boss (Figure 2.6). As the teleport feature has been added to the game, the sprite of the object is also a new one to this game (Figure 2.2 - 2.4).

On the other hand, all other tiles of the game are the old set of resources (Figure 2.7).

Figure 2.5. Player sprite

Figure 2.6. Boss sprite

Figure 2.7. Tile sprite

2.2. Project algorithms

In the beginning, when the team decided to make a pathfinding boss as the main villain of the games, there are some algorithms has been considered to implement, but mainly Dijkstra and A* algorithm.

In comparison, A* is basically an informed variation of Dijkstra which is also considered a "best first search" because it greedily chooses which vertex to explore next, according to the value of f(v) [f(v) = h(v) + g(v)] - where h is the heuristic and g is the cost so far (Figure 2.6). A* is both complete (finds a path if one exists) and optimal (always finds the shortest path) if used with an Admissible heuristic function.

```
public void getCost(Node node) {
    //G cost
    int xDis = Math.abs(node.col - startNode.col);
    int yDis = Math.abs(node.row - startNode.row);
    node.gCost = xDis + yDis;
    //H cost
    xDis = Math.abs(node.col - goalNode.col);
    yDis = Math.abs(node.row - goalNode.row);
    node.hCost = xDis + yDis;
    //F cost
    node.fCost = node.gCost + node.hCost;
}
```

Figure 2.8. The creation of the value of f(v), g(v), h(v)

```
//Find the best node
int bestNodeIndex = 0;
int bestNodeFCost = 999;

for (int i = 0; i < openList.size(); i++) {
    if (openList.get(i).fCost < bestNodeFCost) {
        bestNodeIndex = i;
        bestNodeFCost = openList.get(i).fCost;
    } else if (openList.get(i).fCost == bestNodeFCost) {
        if (openList.get(i).gCost < openList.get(bestNodeIndex).gCost) {
            bestNodeIndex = i;
        }
    }
}
```

Figure 2.9. Find the best node

Therefore, after some considerations between Dijkstra and A* algorithm, the team decide to implement A* algorithm as the one that controls and directs the boss.

3. UML Diagram

Everything happens in the Scene. Scene acts as a menu where it will call the draw function from all objects from map to player. It is also in charge of updating the status of each object. This class is basically the core of our Game Engine (Figure 2.16). While the Window creates a window for the Scene to action on that.

Our main algorithm is A* have 2 classes as Node and PathFinding shown in Figure 2.12. with the appearance of Sound class which, as its name, is in charge of all sound in the game, all of that can be called feature classes.

As entities such as player and boss have some similar properties, we will extend them from the Entity class. (Figure 2.11). For all of the items we make for the game, there will be a SuperObject, and other unique items you want to create will inherit from it (Figure 2.14).

About the graphic, SpriteSheet and BufferedImageLoader are mainly in charge of it, to be more specific, it is used to crop and load images of sprites, respectively (Figure 2.13).

Finally, these classes help us to do calculations, check position, and some keyboard events (Figure 2.10) and the class Constant is here to declare all kinds of Constants Variables common in use (Figure 2.15).

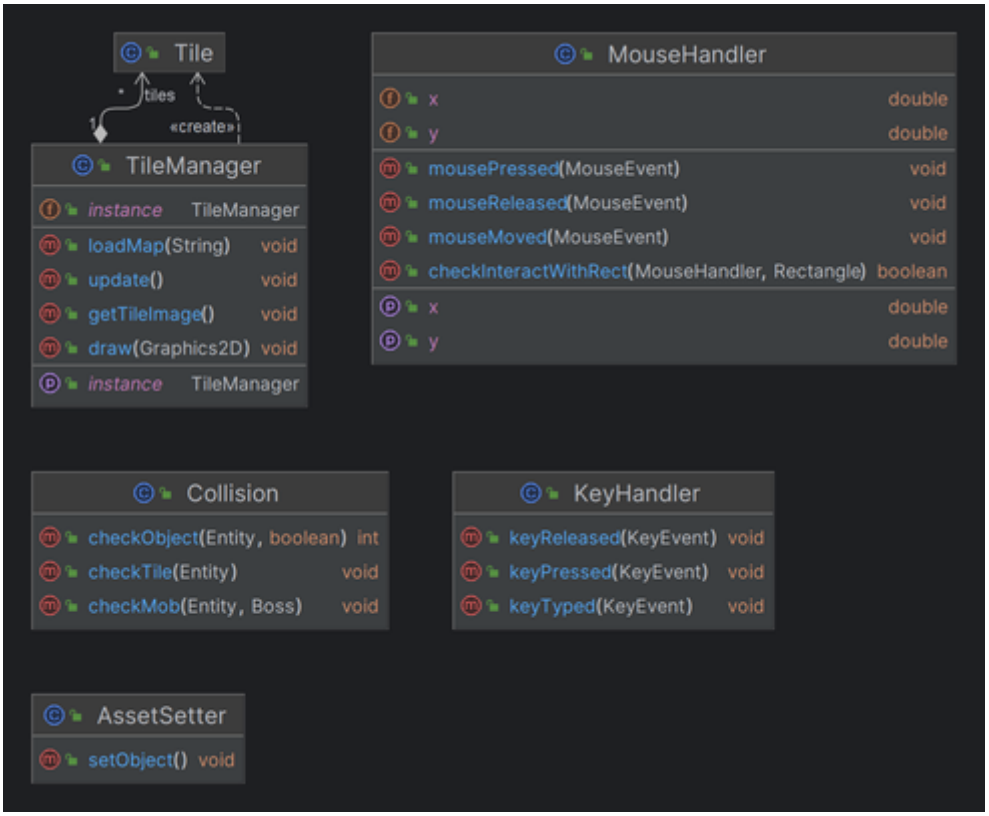

Figure 2.10. Utility classes

Figure 2.11. Entity classes and their inheritors

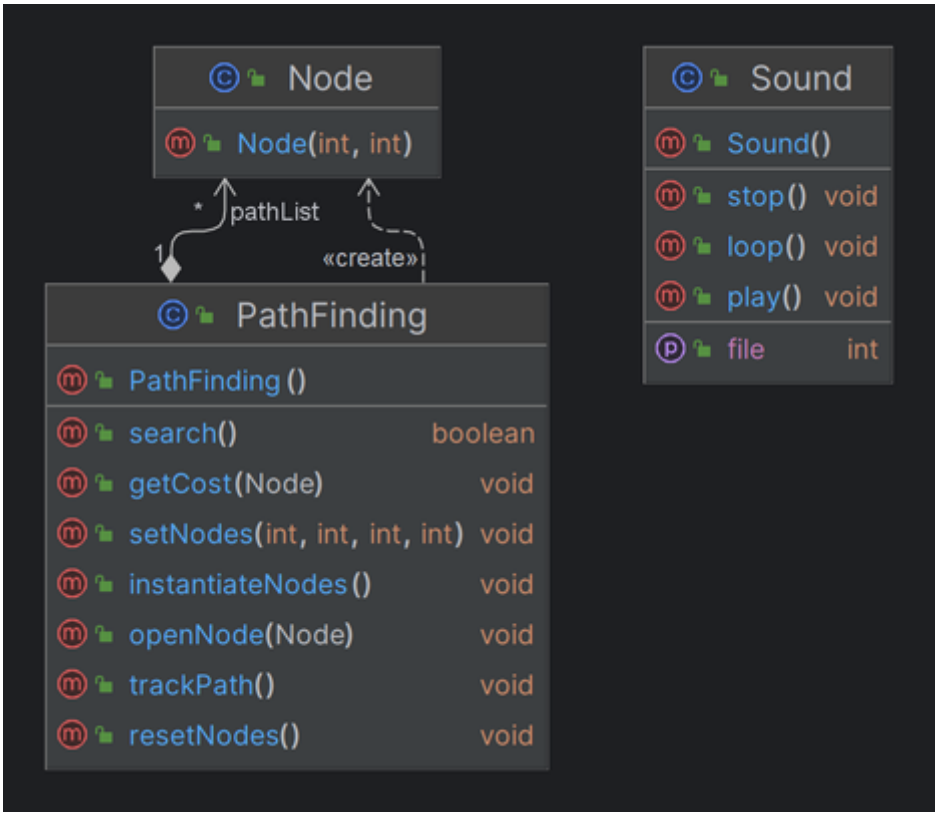

*Figure 2.12. A*algorithm classes and Sound class*

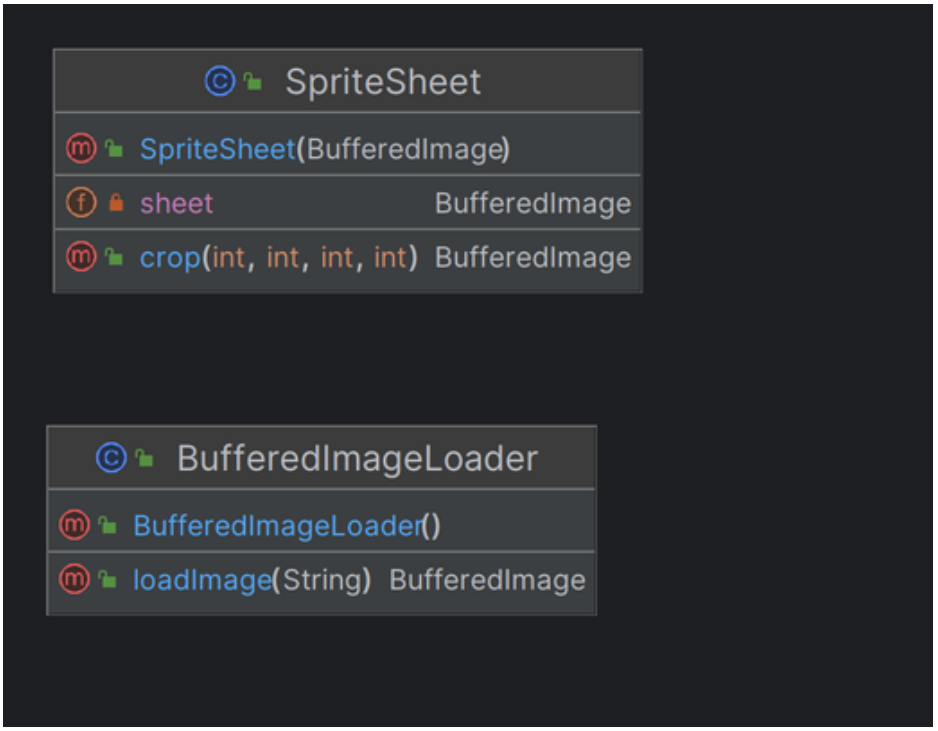

Figure 2.13. Graphics related classes

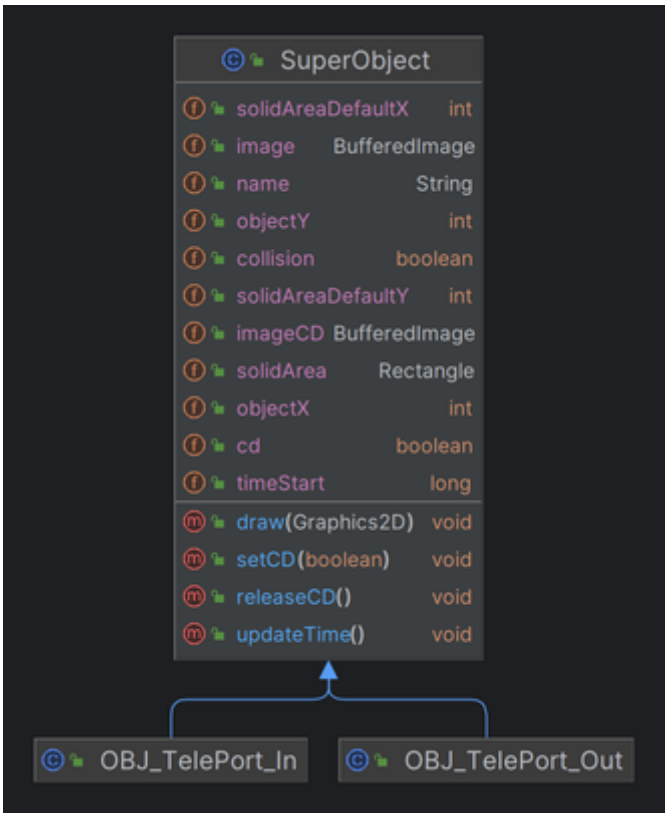

Figure 2.14. Objects class

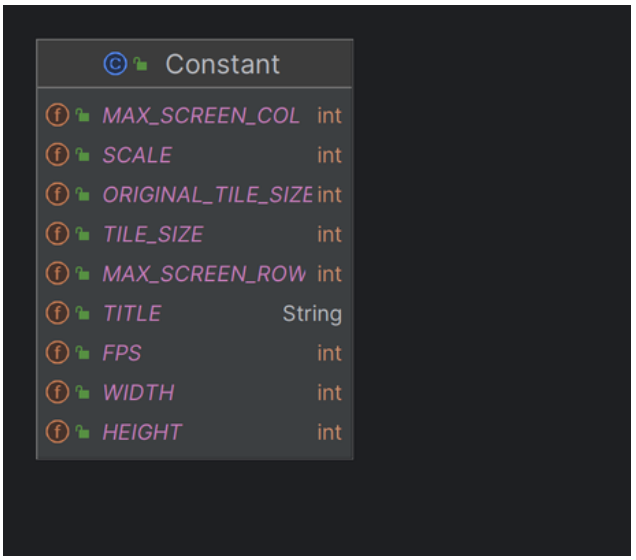

Figure 2.15.Constant class

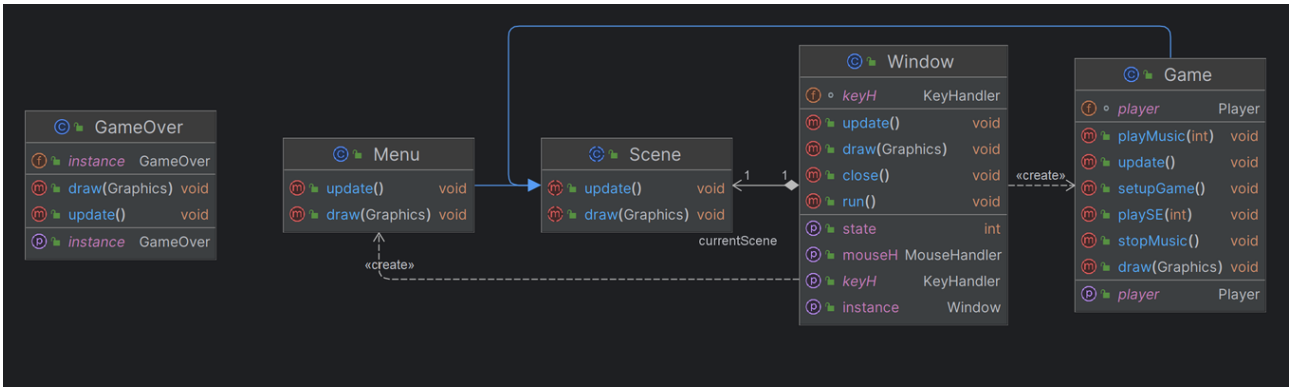

Figure 2.16. Configuration classes

CHAPTER 3. DEMO – RESULT

To run the game, users can install this project from GitHub. Then, users can start to play by running the main class App.java.

After running this class, the game's window will be created with the Menu state appears.

Figure 3.1. The Menu state appears

To start the game, players have to click the button. Then, this will change to the Game state, where players can enjoy the game.

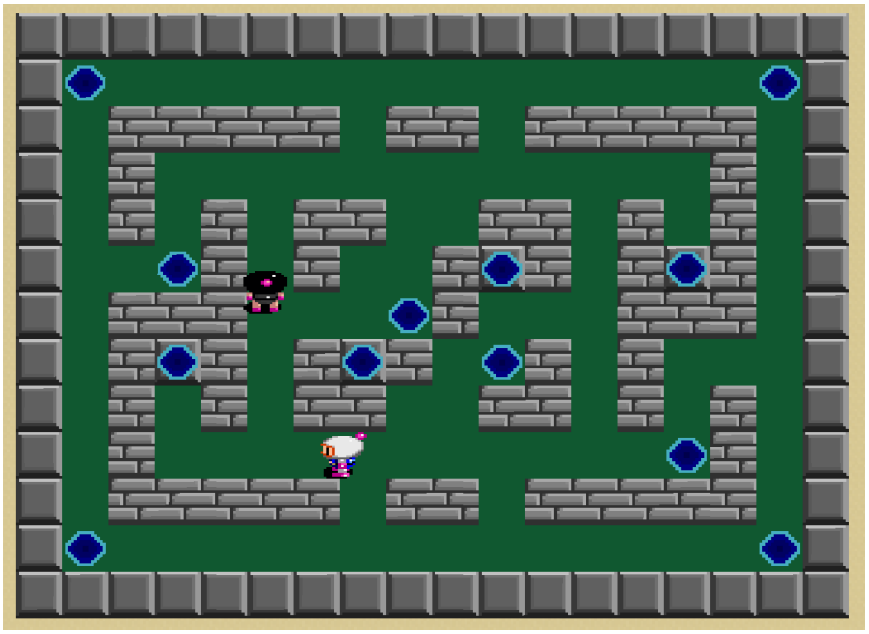

Figure 3.2. The Game state is running

When the boss catches the player, the game will end and the Game Over state will appear. There are 2 options for players to choose: restart the game or exit the game.

Figure 3.3. Game Over state appears

CHAPTER 4. CONCLUSION AND FUTURE WORKS

1. Conclusion

The team has learned more about algorithms and data structures throughout this project, particularly with regard to pathfinding algorithms. In addition, programming and project management skills have been noticeably improved. As a result, the team will benefit from this knowledge and experience for future learning and careers.

2. Future work

Regarding the game, the team hopes to complete the game by adding some necessary features such as the timer or advanced skills for the player and boss. Additionally, the code can be improved for better optimization, which will improve the game's performance.

Regarding algorithms and data structures, the team believes that this knowledge will be applied successfully in our upcoming projects at school and also at future work.

3. Acknowledgment

The team hopes to express the sincerest appreciation to our lecturers who have helped to complete this project: Dr. Tran Thanh Tung and MSc. Pham Quoc Son Lam.

REFERENCES

Java Swing Tutorial - javatpoint. www.javatpoint.com. (n.d.). Retrieved January 7, 2023, from https://www.javatpoint.com/java-swing

Trail: Sound. Trail: Sound (The Java™ Tutorials). (n.d.). Retrieved January 7, 2023, from https://docs.oracle.com/javase/tutorial/sound/index.html

A Search Algorithm.* www.geeksforgeeks.org. (n.d.). Retrieved March 8, 2023, from https://www.geeksforgeeks.org/a-search-algorithm/

Free online sprite editor. Piskel. (n.d.). Retrieved January 7, 2023, from https://www.piskelapp.com/