INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

School of Computer Science and Engineering

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

PROJECT REPORT

Minesweeper

Adviser: Dr Vi Chi Thanh

Course: Data Structure & Algorithm

Semester 2 (2024-2025)

No.	Full Name	Student's ID	Contribution
1	Nguyễn Đức Nguyên Phúc	ITDSIU21108	100%

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Minesweeper is a logic puzzle game based on abstraction and simplification. The goal is to locate mines in a fixed-size grid by flagging or clicking cells without bombs. In case, a

player clicks on a cell containing a bomb, all cells with bombs are revealed, resulting in a loss. Conversely, if a player successfully flags all bombs without clicking on any, they win.

Minesweeper combines strategic thinking and prudent decision-making, offering players an immersive experience as they analyze and predict the position of mines. In this endeavour, I have developed the game "Minesweeper" to provide players with a version suitable for all ages. The game's theme is based on the concept of the original Minesweeper game.

Keywords: Minesweeper, logic, Data Structure & Algorithm.

CHAPTER 1: INTRODUCTION

1. Objectives

The project aims to develop a game that combines puzzle logic and strategy game concepts. It showcases core principles of object-oriented programming, data structures, and algorithms. As a standalone game, it requires analytical thinking and problem-solving skills. Despite its simplicity, the game is accessible to players of all ages, fostering critical thinking.

In Minesweeper, players navigate a pointer to click on cells to reveal safe spaces while avoiding bombs. The game also involves strategic flagging of suspected bomb locations.

In summary, the project's objectives are:

- Create an entertaining and engaging game experience.
- Implement object-oriented programming techniques and data structures/algorithms.
- Enhance the game's management and code efficiency.
- Explore potential features for future expansion.

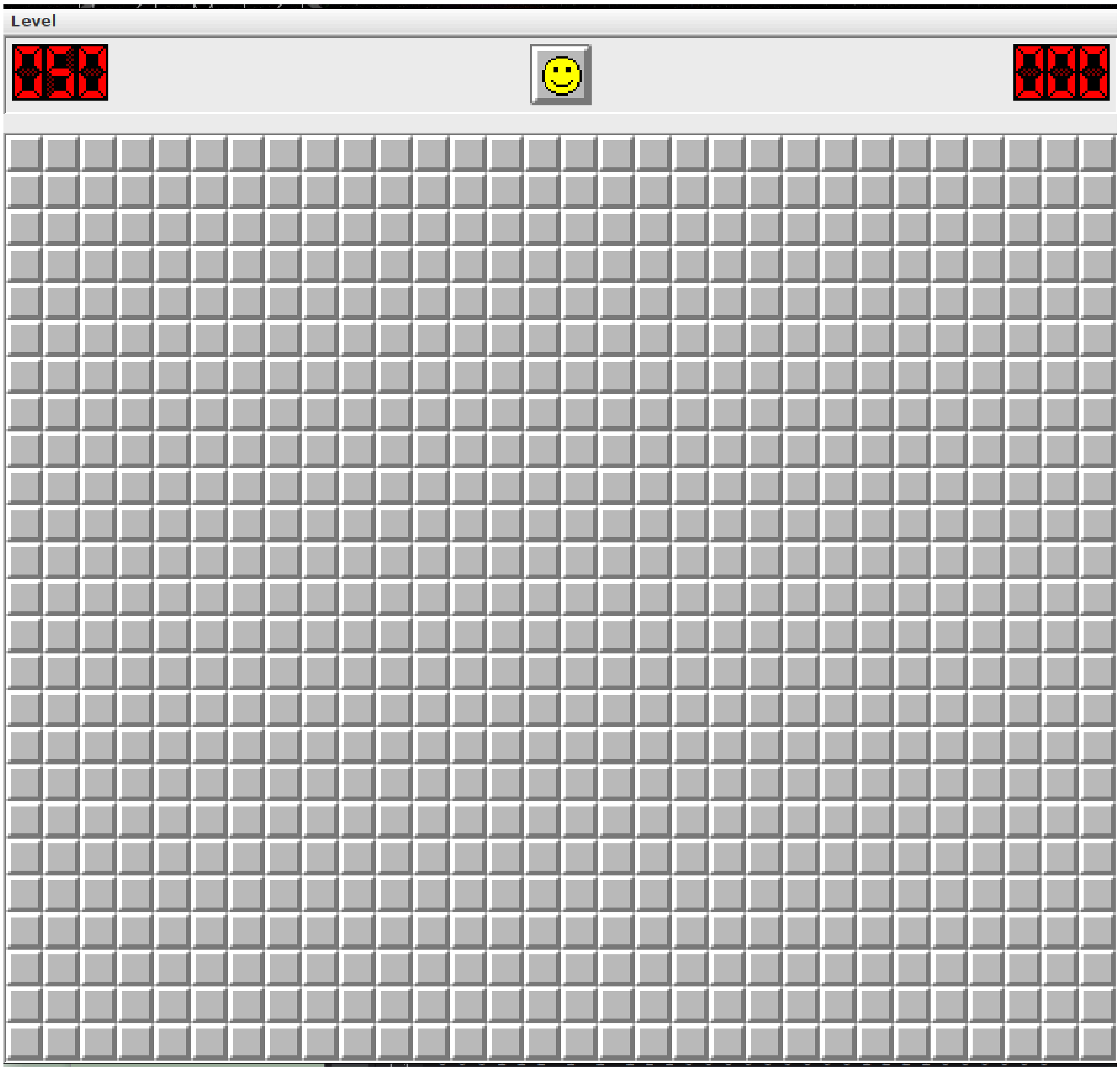


Figure 1. Minesweeper

2. The tools used

- IDE for programming and debugging: IntelliJ IDEA, Eclipse
- Java Development Kit: 21.
- Mean of code version management: GitHub.

3. Git Explanation

- **Repository:** This project is stored in a Git repository. The Git Repository is a repository for source code, change history, and project-related files.
- **Commits**: Each change in the source code is recorded as a commit. Each commit contains the changes made, who made the changes, and a message describing what was changed.
- **Branches**: To manage features and parallel development, this project can use the *main* branch to contain all project changes
- **GitHub**: This project can be hosted on the Git platform GitHub. GitHub offers features like repository management, tracking changes, creating pull requests, and collaborating on projects.
- **Git Statistics**: As shown in Figure 2, Git provides project statistics, such as the number of commits, and other activities. This information helps track and manage project development progress.

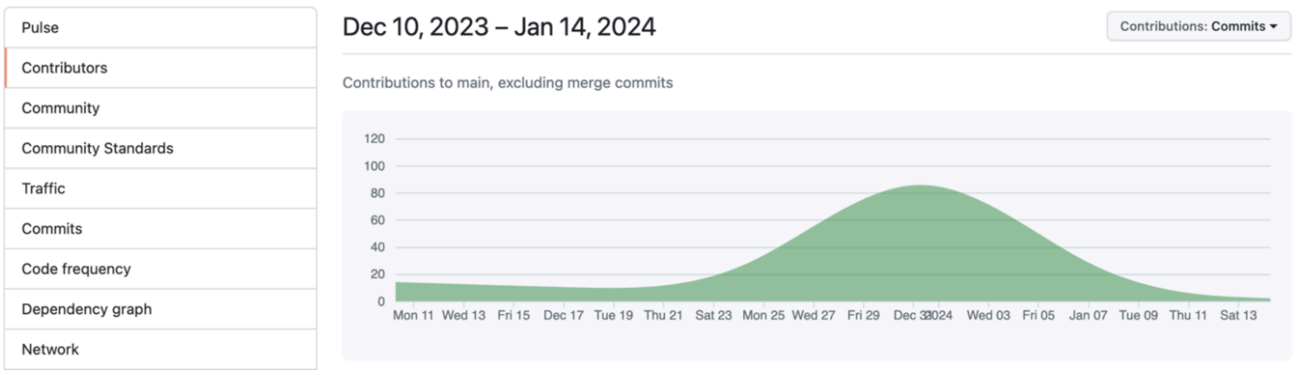


Figure 2. GitHub statistics

In short, Git is used to manage source code, track changes, and collaborate during the development of this "Minesweeper" project. Git's features and workflows help ensure transparency, tractability, and effective collaboration among team members.

4. Feature:

- Interactive User Interface: Observer and Strategy pattern updates in real-time.
- Changeable Game Logic: Easily transition between various setups and tactics.
- Effective Cell Exploration: BFS and recursive algorithms were used to achieve the best possible results.
- Media management: Stores and retrieves audio and picture files fast using a hashtable.

CHAPTER 2: METHODOLOGY

1. Rules

Key Rules of the Game:

When clicking on a cell:

- If it contains a bomb, you lose.
- If it's safe, a number appears, indicating the number of bombs surrounding it.

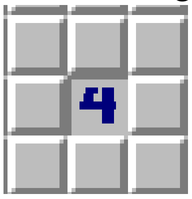


Figure 3. Example

- For instance: If you click on a cell with the value 4, it indicates that there are four randomly placed bombs in the adjacent cells.

2. Design

a. *UI/UX*

Prioritizing UI/UX before our game's launch enabled us to gather valuable feedback from external sources, guiding us in establishing a consistent design pattern.

Minesweeper is a strategy and puzzle game that challenges players to locate hidden mines on a game board. The game requires a combination of strategic thinking and luck to succeed.

In this game, players click on unrevealed empty squares. When a square is clicked, several outcomes are possible:

- In case, no surrounding cells contain mines, the square is revealed as blank.
- In case, surrounding cells contain mines, the square displays a number indicating the number of mines adjacent to it.
- In case, a player clicks on a cell containing an unrevealed mine, it converts to a revealed mine, and all unrevealed mines on the board are shown. Consequently, the game is lost.
- In case, you suspect that a cell contains a hidden mine, you can flag it to prevent accidentally clicking on it.

 It is essential to avoid clicking on cells that contain unrevealed mines.

Crucial Components for Minesweeper UI/UX:

- Information to display:

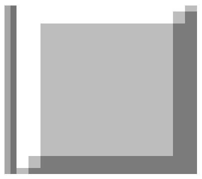


Figure 4. An unrevealed empty square

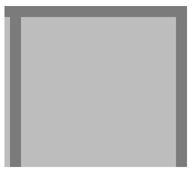


Figure 5. A revealed as blank

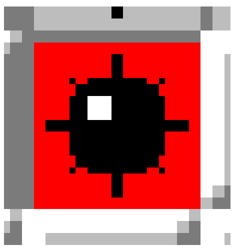


Figure 6: a revealed mine

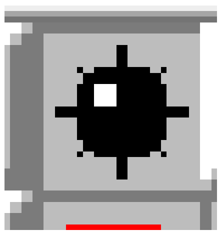


Figure 7: an unrevealed mine

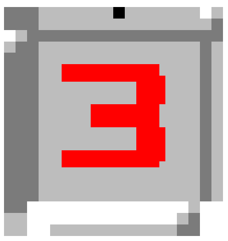
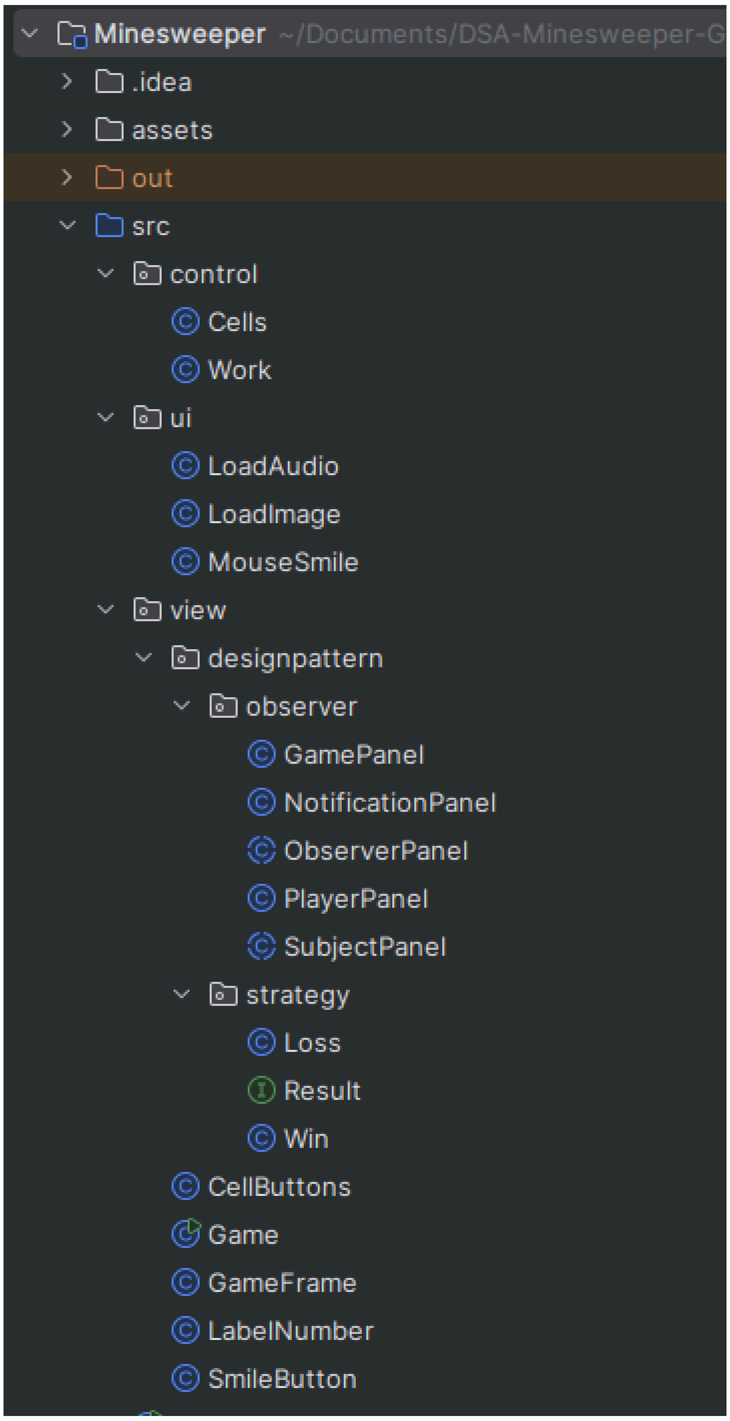


Figure 8: Digits ('1'-'8')

b. Game algorithm

After much iteration and troubleshooting, we now have the project structure as seen in the image below.



```
Minesweeper ~/Documents/DSA-Minesweeper-G
  .idea
  assets
  out
  src
    control
      Cells
      Work
    ui
      LoadAudio
      LoadImage
      MouseSmile
    view
      designpattern
        observer
          GamePanel
          NotificationPanel
          ObserverPanel
          PlayerPanel
          SubjectPanel
        strategy
          Loss
          Result
          Win
      CellButtons
      Game
      GameFrame
      LabelNumber
      SmileButton
```

Figure 9. Project structure

The classes we teach can be arranged into distinct groups, like:

- UI: is a package to load audio, and images and control the mouse so it contains the LoadAduio class, LoadImage class and MouseSimle class
- Game: This is the main class of the game program. It contains the main() method and is used to initialize and run the game.
- Cell Buttons: This is a class that represents cells in the game. It can be displayed and interacted with by the user via buttons.
- Cells: This is a simple class that represents a cell in the game's matrix. It contains the (x, y) coordinates of that cell in the matrix.
- Game Frame: This is a class that represents the main window of the game. It may contain user interface elements such as buttons, labels, and other elements to form the game interface. This class can have methods and properties to create and manage windows, performing actions when the user interacts with the interface.
- Label Number: This is a class that represents a label in the game. This label can contain and display a number, information or status in the game. This class can provide methods to set and update the value of the label.
- Smile Button: This is a class that represents a smile button in the game. Commonly found in land mine games, the smile button is used to represent the game's mood state (e.g. happy, sad, worried). This class may have methods to change the smile button image based on the state of the game or user interaction.

Apply design pattern (Observer and Strategy design pattern):

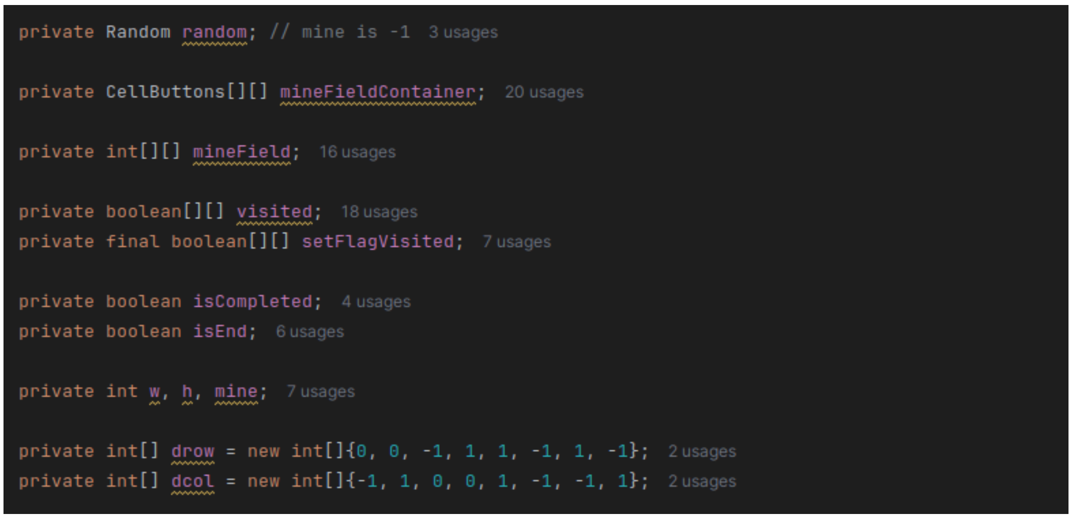
- Observer design pattern:
  - SubjectPanel: This is an abstract class inherited from the Panel class. This class represents a user interface element and is a "subject" in the Observer design model. It contains a list of ObserverPanels and provides methods to manage and notify ObserverPanels when there is a change in their state.
  - PlayerPanel: This class inherits from ObserverPanel and represents a specific user interface element. It contains a matrix of CellButtons and is used to display

cells in the game. It also implements the binding of mouse events to the cells and binds to the SubjectPanel to receive notifications of changes.

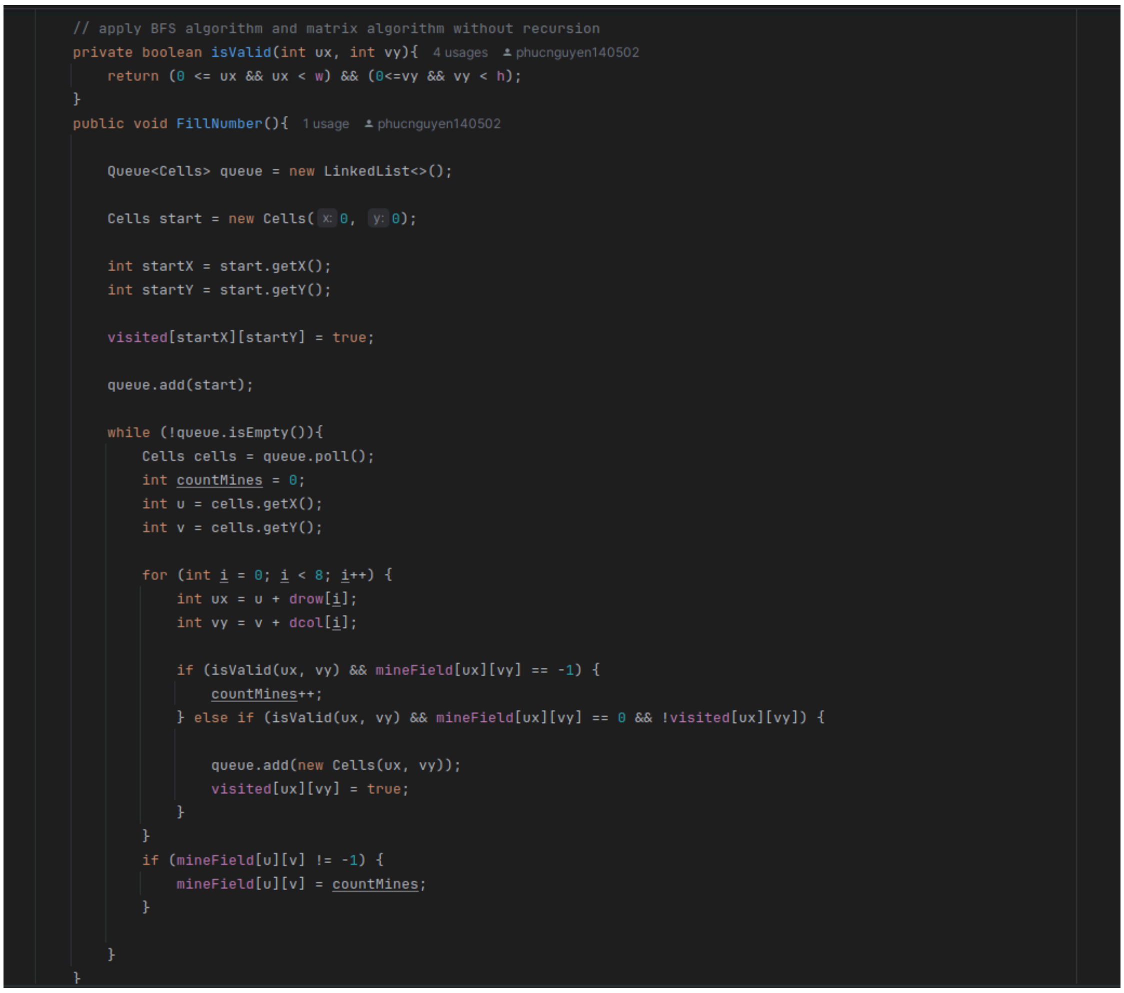
- ObserverPanel: This abstract class inherits from the JPanel class and represents an Observer in the Observer model. It is connected to a specific SubjectPanel and will receive notifications when there is a change in the SubjectPanel's state.
- NotificationPanel: This class inherits from ObserverPanel and represents a specific user interface element. It contains elements such as time, smileys, and remaining mines. It also provides methods to update these values based on the state of the SubjectPanel and display them on the interface.
- GamePanel: This class inherits from SubjectPanel and also implements the MouseListener interface. It represents a specific user interface component and contains other components such as NotificationPanel and PlayerPanel. It also handles mouse events and interacts with Work (a class not defined in the documentation) to perform in-game actions.

- Strategy design pattern:
  - Result: This is an interface that defines the
  - Loss: This class implements the Result interface and represents the loss result in the game. When the game result is a loss, the outCome() method will be called to handle related actions such as displaying a message, replaying the game, or ending the game.
  - Win: This class implements the Result interface and represents the winning result in the game. When the game result is a win, the outCome() method will be called to handle related actions such as displaying a message, replaying the game, or ending the game.
- Generic:
  - To build a code base that is both versatile and type-safe, generics are heavily utilized. This method guarantees that the code is flexible enough to accommodate many game settings and tactics, in addition to being reliable.

Apply Data Structure & Algorithm:

- Work: This is the class responsible for handling game logic. It includes methods for checking cell status, handling events when users click on cells, and calculating numbers for cells.
    - Advanced Graph (Breadth-First Search (BFS)) and Matrix:
        - 

```java
private Random random; // mine is -1  3 usages

private CellButtons[][] mineFieldContainer;  20 usages

private int[][] mineField;  16 usages

private boolean[][] visited;  18 usages
private final boolean[][] setFlagVisited;  7 usages

private boolean isCompleted;  4 usages
private boolean isEnd;  6 usages

private int w, h, mine;  7 usages

private int[] drow = new int[]{0, 0, -1, 1, 1, -1, 1, -1};  2 usages
private int[] dcol = new int[]{-1, 1, 0, 0, 1, -1, -1, 1};  2 usages
```



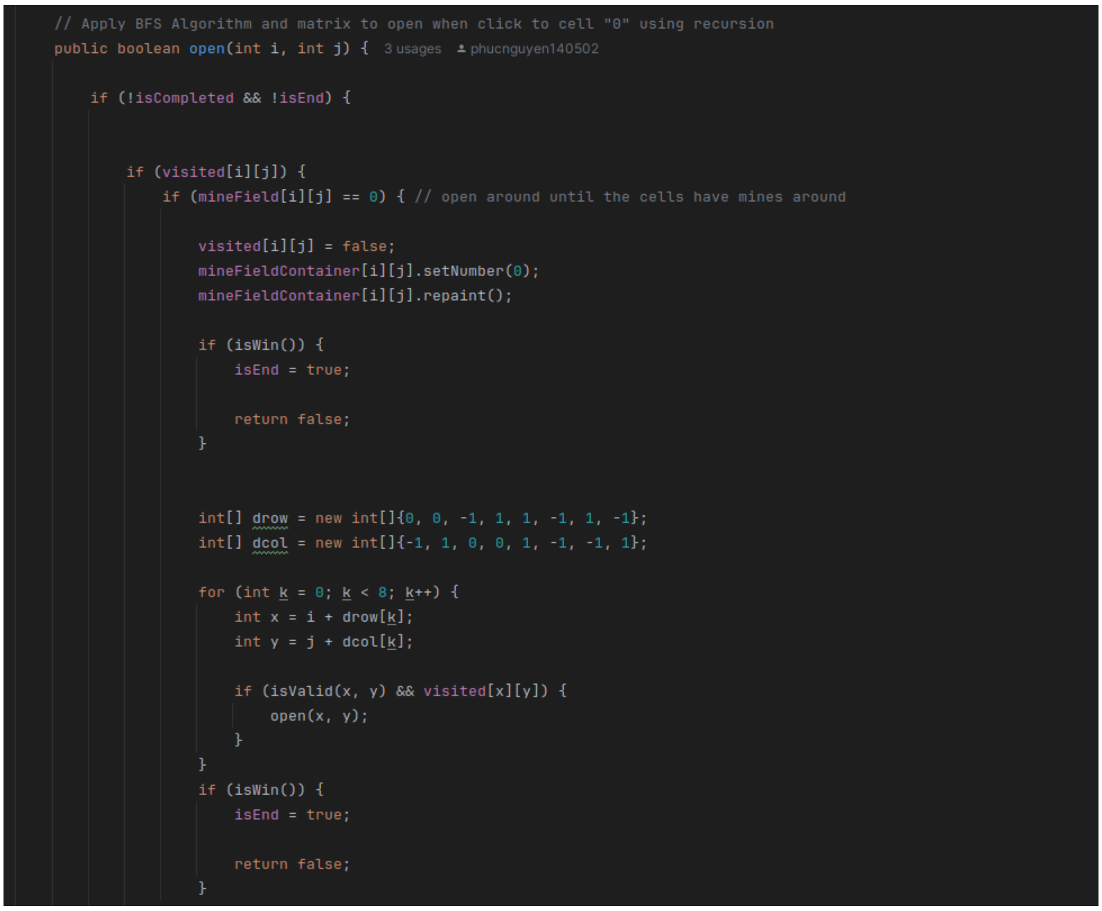
```
// apply BFS algorithm and matrix algorithm without recursion
private boolean isValid(int ux, int vy){  4 usages  phucnguyen140502
    return (0 <= ux && ux < w) && (0<=vy && vy < h);
}
public void FillNumber(){  1 usage  phucnguyen140502

    Queue<Cells> queue = new LinkedList<>();

    Cells start = new Cells( x: 0,  y: 0);

    int startX = start.getX();
    int startY = start.getY();

    visited[startX][startY] = true;

    queue.add(start);

    while (!queue.isEmpty()){
        Cells cells = queue.poll();
        int countMines = 0;
        int u = cells.getX();
        int v = cells.getY();

        for (int i = 0; i < 8; i++) {
            int ux = u + drow[i];
            int vy = v + dcol[i];

            if (isValid(ux, vy) && mineField[ux][vy] == -1) {
                countMines++;
            } else if (isValid(ux, vy) && mineField[ux][vy] == 0 && !visited[ux][vy]) {

                queue.add(new Cells(ux, vy));
                visited[ux][vy] = true;
            }
        }
        if (mineField[u][v] != -1) {
            mineField[u][v] = countMines;
        }

    }
}
```

Figure 10. Apply Advanced Graph (BFS)

--> The code in the Java code fills the number of adjacent mines in a minefield using a queue. It creates a queue and enters a loop to check if a cell is within the minefield and contains a mine. If it does, the count of adjacent mines is incremented. If not, it is added to the queue and marked as visited. The process continues until all cells in the minefield have been visited and their number of adjacent mines is calculated. The code has a time complexity of O(n^2) and a space complexity of O(n^2).

Time Complexity: O(n^2)

- Recursion:



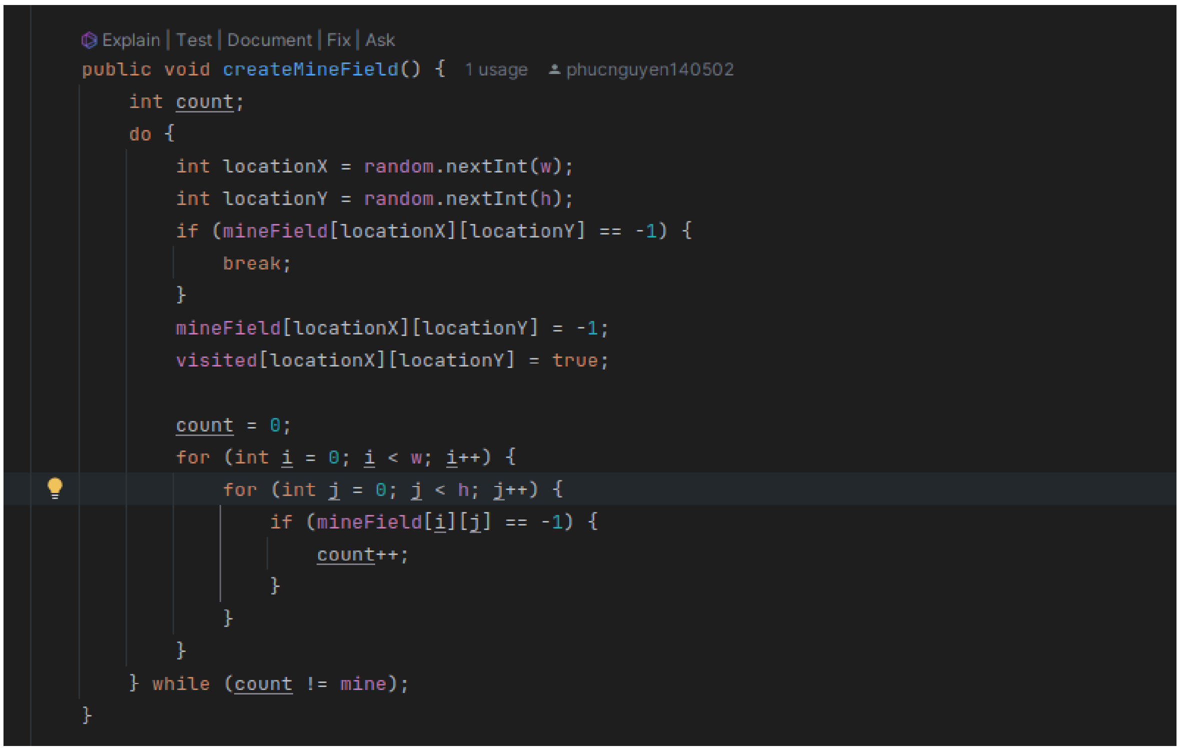
```java
// Apply BFS Algorithm and matrix to open when click to cell "0" using recursion
public boolean open(int i, int j) {  3 usages  ≛ phucnguyen140502

    if (!isCompleted && !isEnd) {


        if (visited[i][j]) {
            if (mineField[i][j] == 0) { // open around until the cells have mines around

                visited[i][j] = false;
                mineFieldContainer[i][j].setNumber(0);
                mineFieldContainer[i][j].repaint();

                if (isWin()) {
                    isEnd = true;

                    return false;
                }


                int[] drow = new int[]{0, 0, -1, 1, 1, -1, 1, -1};
                int[] dcol = new int[]{-1, 1, 0, 0, 1, -1, -1, 1};

                for (int k = 0; k < 8; k++) {
                    int x = i + drow[k];
                    int y = j + dcol[k];

                    if (isValid(x, y) && visited[x][y]) {
                        open(x, y);
                    }
                }
                if (isWin()) {
                    isEnd = true;

                    return false;
                }
```

Figure 11. Apply Recursion

--> The code in the Minesweeper project is part of the Work class, managing game logic, creating the minefield, filling adjacent mines, and handling user interactions. The open method is crucial for revealing cell content. It checks if the game is completed and ended, verifies if a cell has been visited, and if not, reveals its content. If the cell contains a mine, it sets its appearance to a ″boom″ image and marks the game as completed. If it doesn’t, it reveals its content, updates its appearance, or toggles the flag state. The code has a time complexity of O(n^2) and a space complexity of O(n^2) due to nested loops.

Time Complexity: O(n^2)

- Create the board:



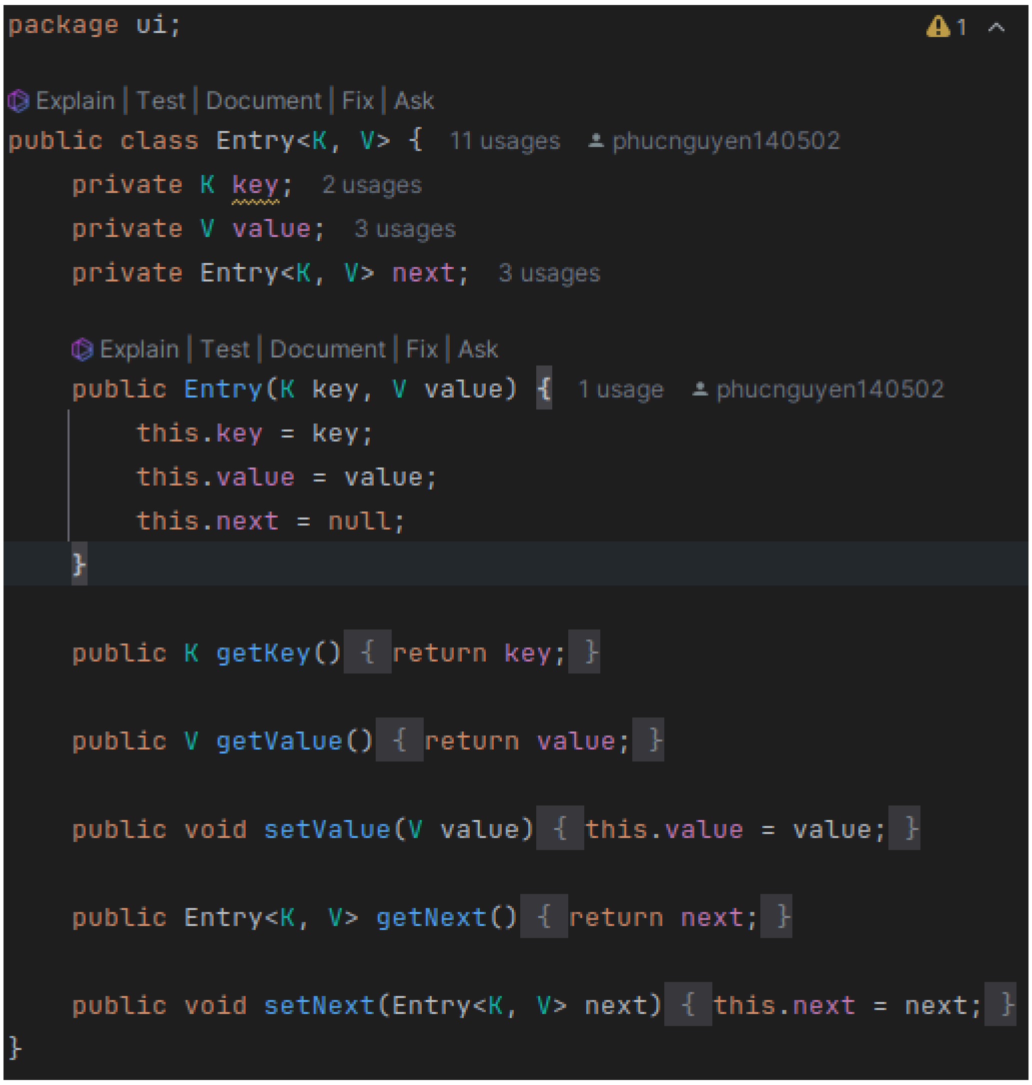
```
public void createMineField() {
    int count;
    do {
        int locationX = random.nextInt(w);
        int locationY = random.nextInt(h);
        if (mineField[locationX][locationY] == -1) {
            break;
        }
        mineField[locationX][locationY] = -1;
        visited[locationX][locationY] = true;

        count = 0;
        for (int i = 0; i < w; i++) {
            for (int j = 0; j < h; j++) {
                if (mineField[i][j] == -1) {
                    count++;
                }
            }
        }
    } while (count != mine);
}
```

Figure 12. Create The Board

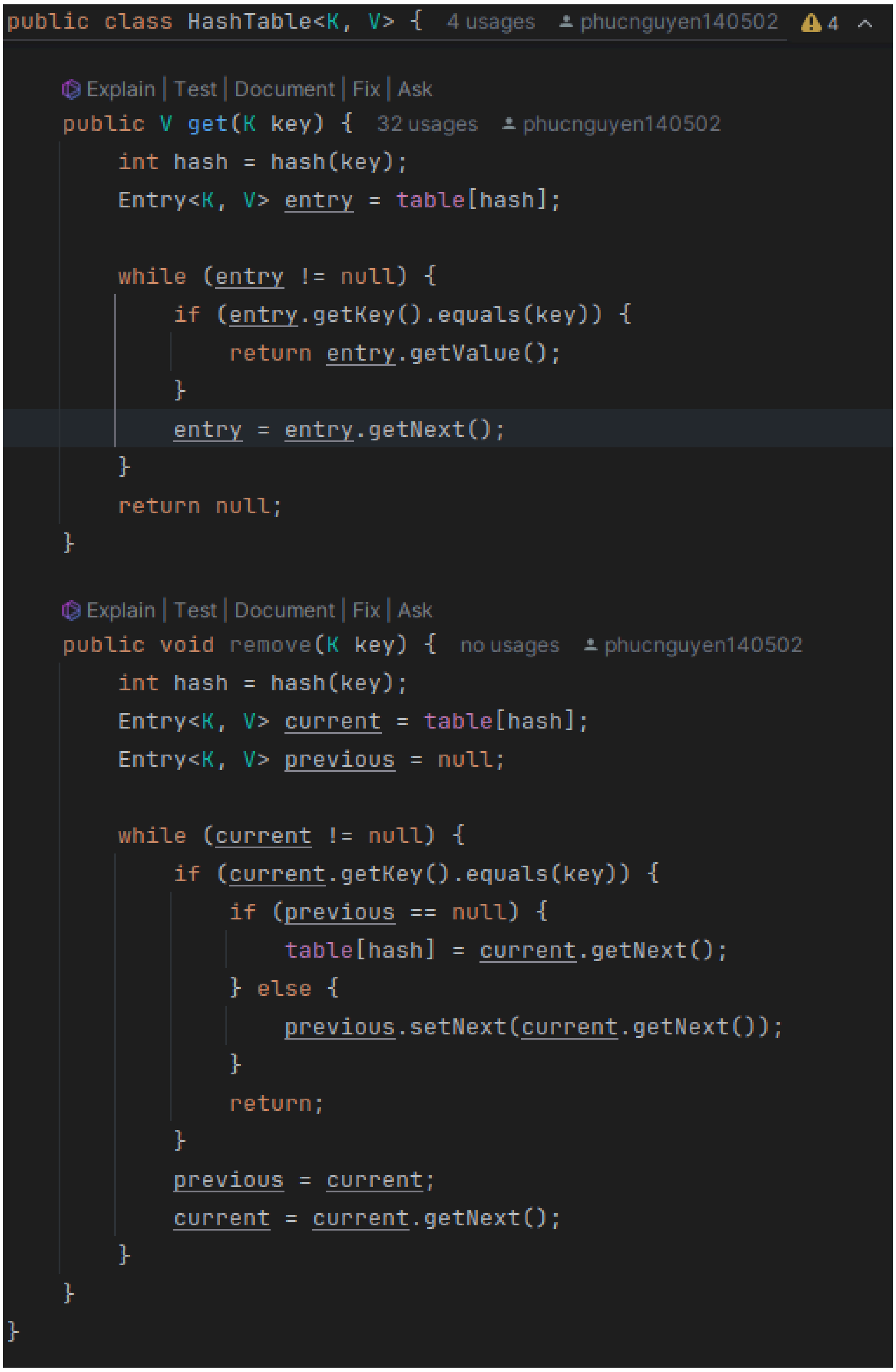
--> The code in the Minesweeper project is responsible for creating a 2D grid representing the game board, known as the minefield. The createMineField method uses a random number generator to place mines in the grid, ensuring the total number of mines is equal to the specified parameter. The code initializes a count variable to 0, then enters a loop that checks if there is a mine at a specific location. If there is, it moves on to the next iteration. If not, it places a mine and marks the corresponding cell in the visited array as true. The method then loops through all the cells in the grid, checking if there is a mine at that location. If there is, it increments the count variable by 1, and if not, it continues to the next iteration. The process continues until the total number of mines in the grid is equal to the specified parameter. The createMineField method has a time complexity of O(w*h), which is directly proportional to the grid's size. The space complexity is O(1), as it only uses a constant amount of additional space to store the count variable and the random object.

Time Complexity: O(w*h)

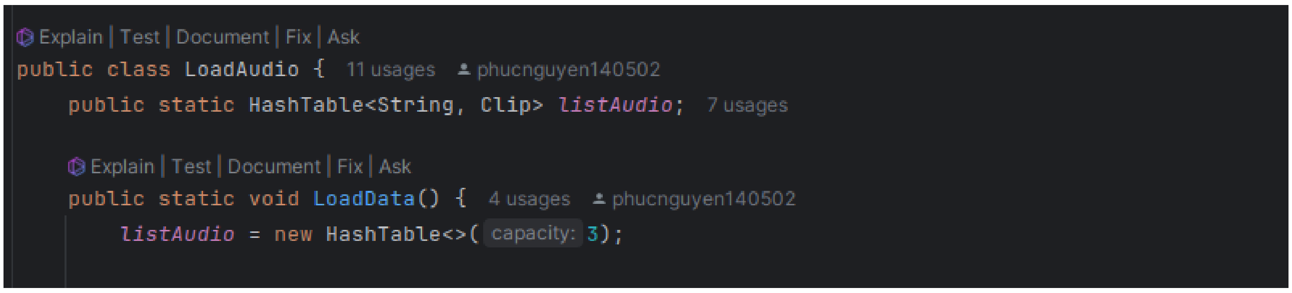
- Hash Table:
  - ■ HashTable and Entry: There are classes to implement the hashtable algorithm, I use the hashtable algorithm to store the audio and image



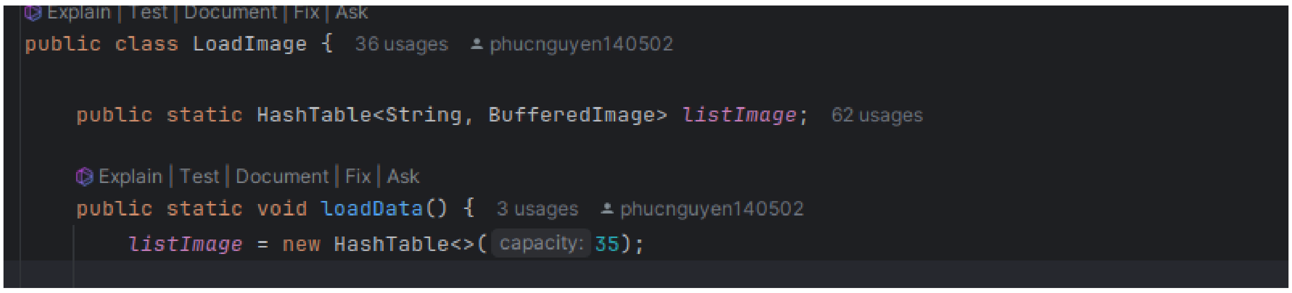
```
package ui;                                                    ⚠1 ^

Explain | Test | Document | Fix | Ask
public class Entry<K, V> {  11 usages  phucnguyen140502
    private K key;   2 usages
    private V value;   3 usages
    private Entry<K, V> next;   3 usages

    Explain | Test | Document | Fix | Ask
    public Entry(K key, V value) {  1 usage  phucnguyen140502
        this.key = key;
        this.value = value;
        this.next = null;
    }

    public K getKey() { return key; }

    public V getValue() { return value; }

    public void setValue(V value) { this.value = value; }

    public Entry<K, V> getNext() { return next; }

    public void setNext(Entry<K, V> next) { this.next = next; }
}
```



```
public class HashTable<K, V> {  4 usages  phucnguyen140502  ⚠4

    Explain | Test | Document | Fix | Ask
    public V get(K key) {  32 usages  phucnguyen140502
        int hash = hash(key);
        Entry<K, V> entry = table[hash];

        while (entry != null) {
            if (entry.getKey().equals(key)) {
                return entry.getValue();
            }
            entry = entry.getNext();
        }
        return null;
    }

    Explain | Test | Document | Fix | Ask
    public void remove(K key) {  no usages  phucnguyen140502
        int hash = hash(key);
        Entry<K, V> current = table[hash];
        Entry<K, V> previous = null;

        while (current != null) {
            if (current.getKey().equals(key)) {
                if (previous == null) {
                    table[hash] = current.getNext();
                } else {
                    previous.setNext(current.getNext());
                }
                return;
            }
            previous = current;
            current = current.getNext();
        }
    }
}
```



```
Explain | Test | Document | Fix | Ask
public class LoadAudio {  11 usages  phucnguyen140502
    public static HashTable<String, Clip> listAudio;  7 usages

    Explain | Test | Document | Fix | Ask
    public static void LoadData() {  4 usages  phucnguyen140502
        listAudio = new HashTable<>( capacity: 3);
```



```
Explain | Test | Document | Fix | Ask
public class LoadImage {  36 usages  phucnguyen140502

    public static HashTable<String, BufferedImage> listImage;  62 usages

    Explain | Test | Document | Fix | Ask
    public static void loadData() {  3 usages  phucnguyen140502
        listImage = new HashTable<>( capacity: 35);
```

Figure 13. Apply HashTable

- The code defines a HashTable class that implements a basic hash table data structure, allowing for the insertion of key-value pairs into the hash table and retrieving the associated value. The class is defined with type parameters K and V for key and value types, and has a private Entry array called table and an integer variable capacity. The constructor HashTable(int capacity) initializes the capacity and creates a new Entry array.

- The hash(K key) method calculates the hash value for a given key using the key's hashCode and the table's capacity. The put(K key, V value) method inserts a new key-value pair into the hash table, either adding it to the table at the calculated index or appending it to the linked list at that index if there is a collision.

- The get(K key) method retrieves the value associated with a given key from the hash table. The time complexity of both put() and get() operations in this HashTable implementation depends on the efficiency of the hash function and the handling of collisions. The time complexity can be considered O(1) on average, but can degrade to O(n) in the worst case scenario where all keys hash to the same index and form a long linked list.

3. UML Diagram

We provided the UML diagrams for the entire project and each group that was addressed to help you better understand the structure and methods.

- Whole UML Diagram: Full UML Diagram, access to view the whole UML Diagram.

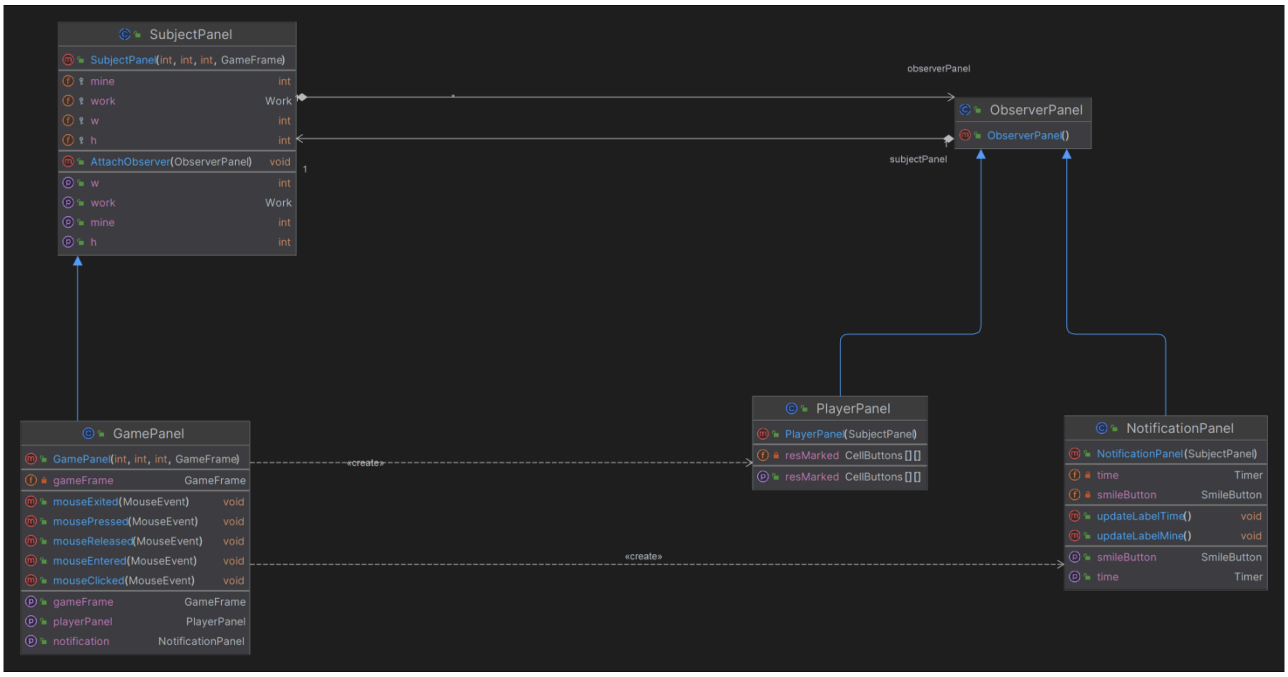

SubjectPanel
SubjectPanel(int, int, int, GameFrame)
mine int
work Work
w int
h int
AttachObserver(ObserverPanel) void
w int
work Work
mine int
h int
observerPanel
subjectPanel
1
ObserverPanel
ObserverPanel()
GamePanel
GamePanel(int, int, int, GameFrame)
gameFrame GameFrame
mouseExited(MouseEvent) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void
mouseEntered(MouseEvent) void
mouseClicked(MouseEvent) void
gameFrame GameFrame
playerPanel PlayerPanel
notification NotificationPanel
«create»
«create»
PlayerPanel
PlayerPanel(SubjectPanel)
resMarked CellButtons[][]
resMarked CellButtons[][]
NotificationPanel
NotificationPanel(SubjectPanel)
time Timer
smileButton SmileButton
updateLabelTime() void
updateLabelMine() void
smileButton SmileButton
time Timer

Figure 14. Observer design pattern

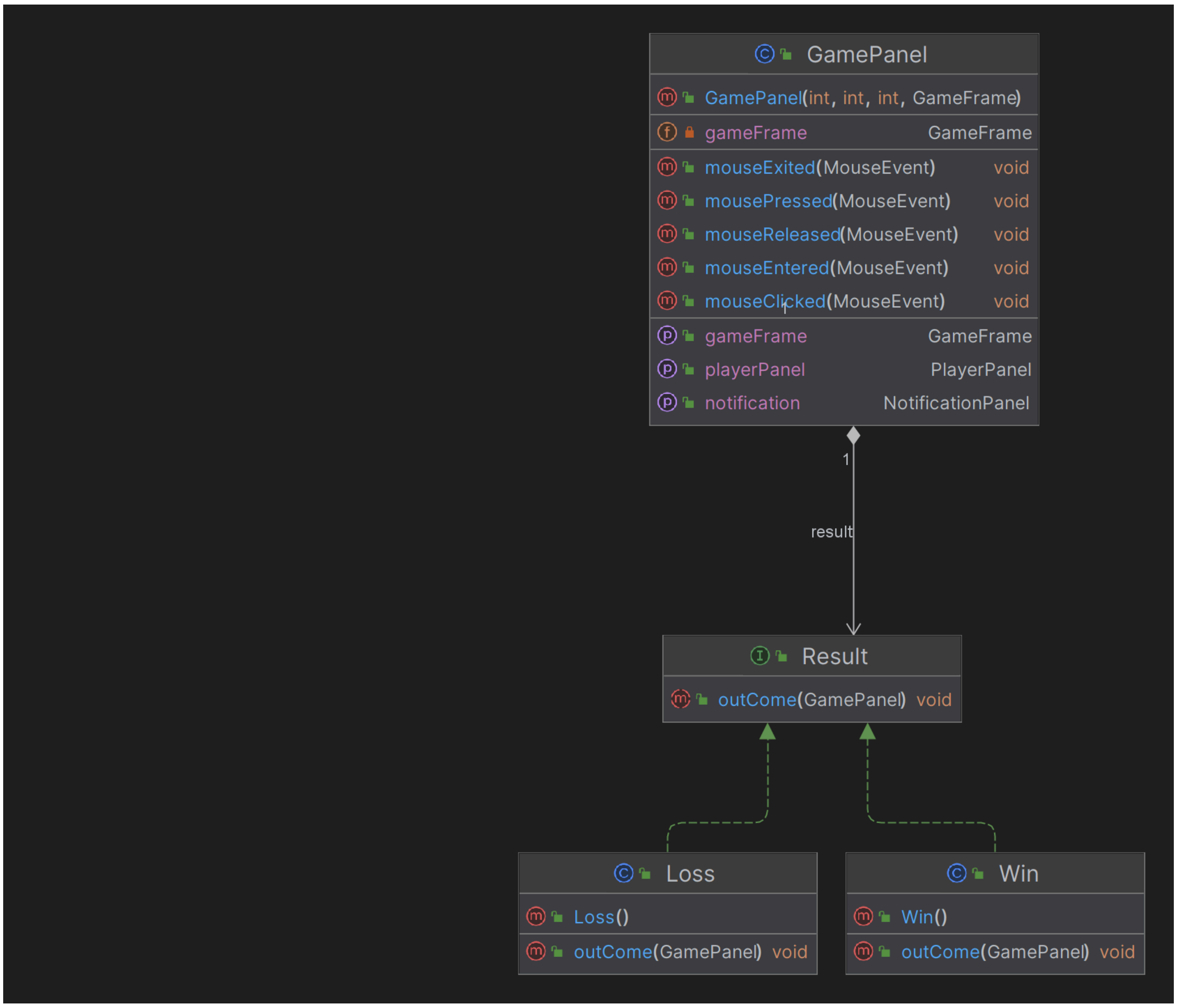

GamePanel
GamePanel(int, int, int, GameFrame)
gameFrame GameFrame
mouseExited(MouseEvent) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void
mouseEntered(MouseEvent) void
mouseClicked(MouseEvent) void
gameFrame GameFrame
playerPanel PlayerPanel
notification NotificationPanel
1
result
Result
outCome(GamePanel) void
Loss
Loss()
outCome(GamePanel) void
Win
Win()
outCome(GamePanel) void

Figure 15. Strategy design pattern

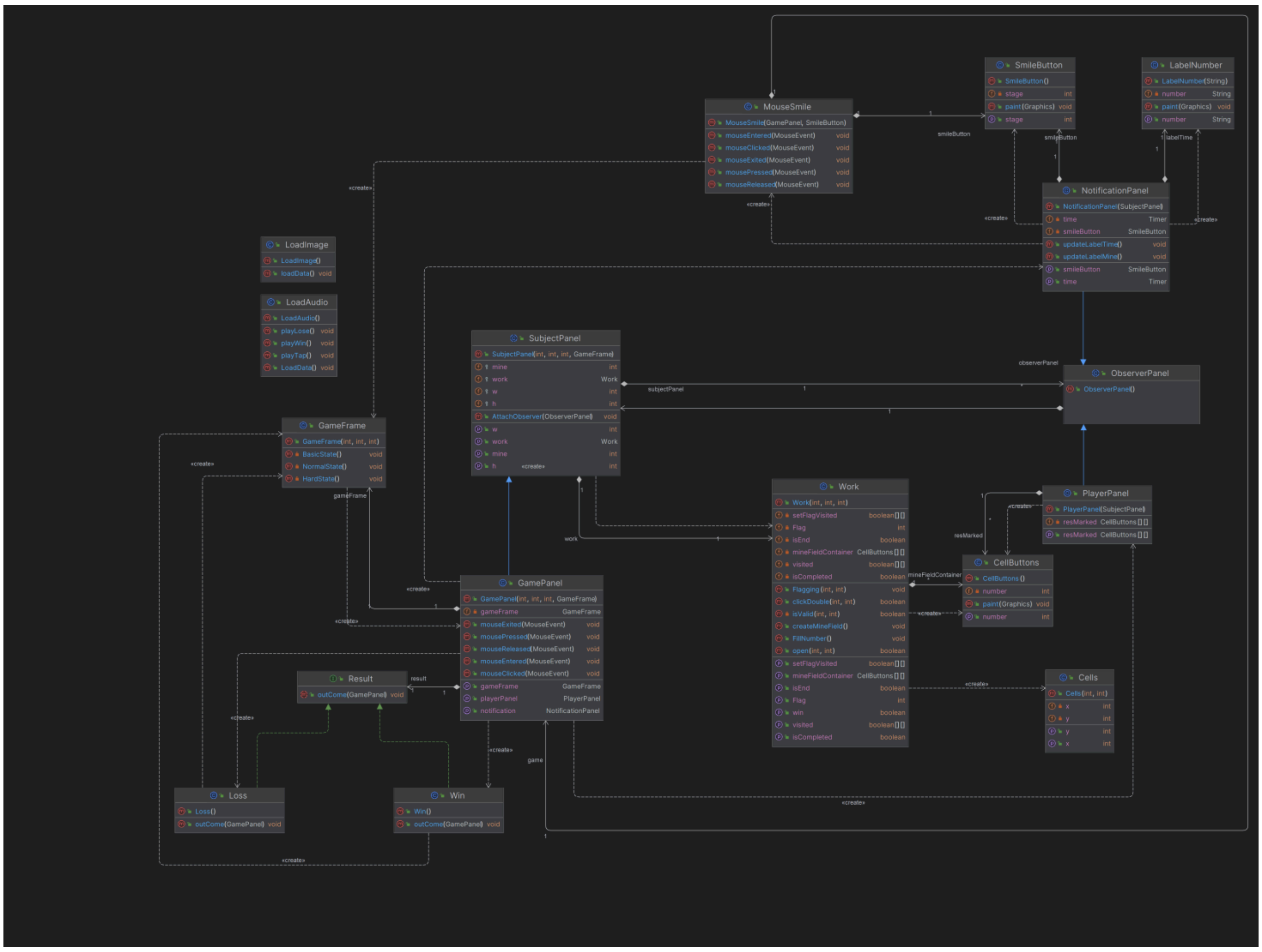


Figure 16. A whole UML diagram

CHAPTER 3: DEMO – RESULT

To test our game on a machine that had an IDE and Java Development Kit 21 installed. We pulled our git repository and ran the Main class to launch the game. The following screenshot shows some examples of the game states in the current build.

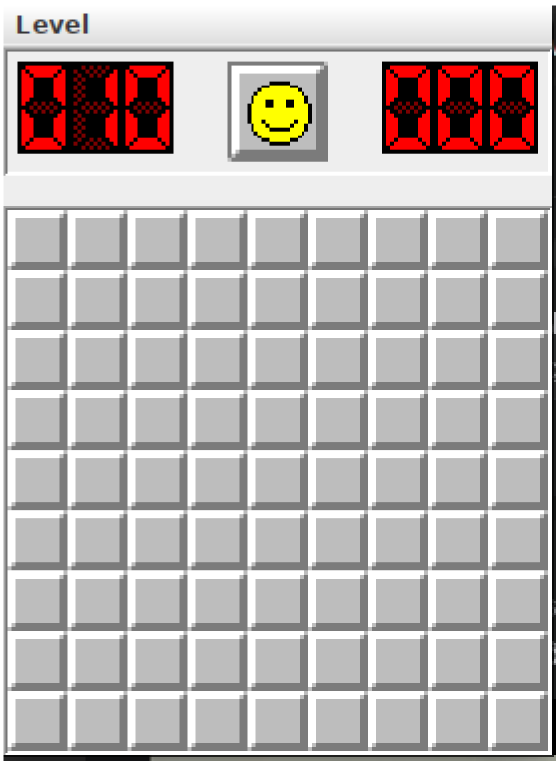


Figure 17. Beginner Level

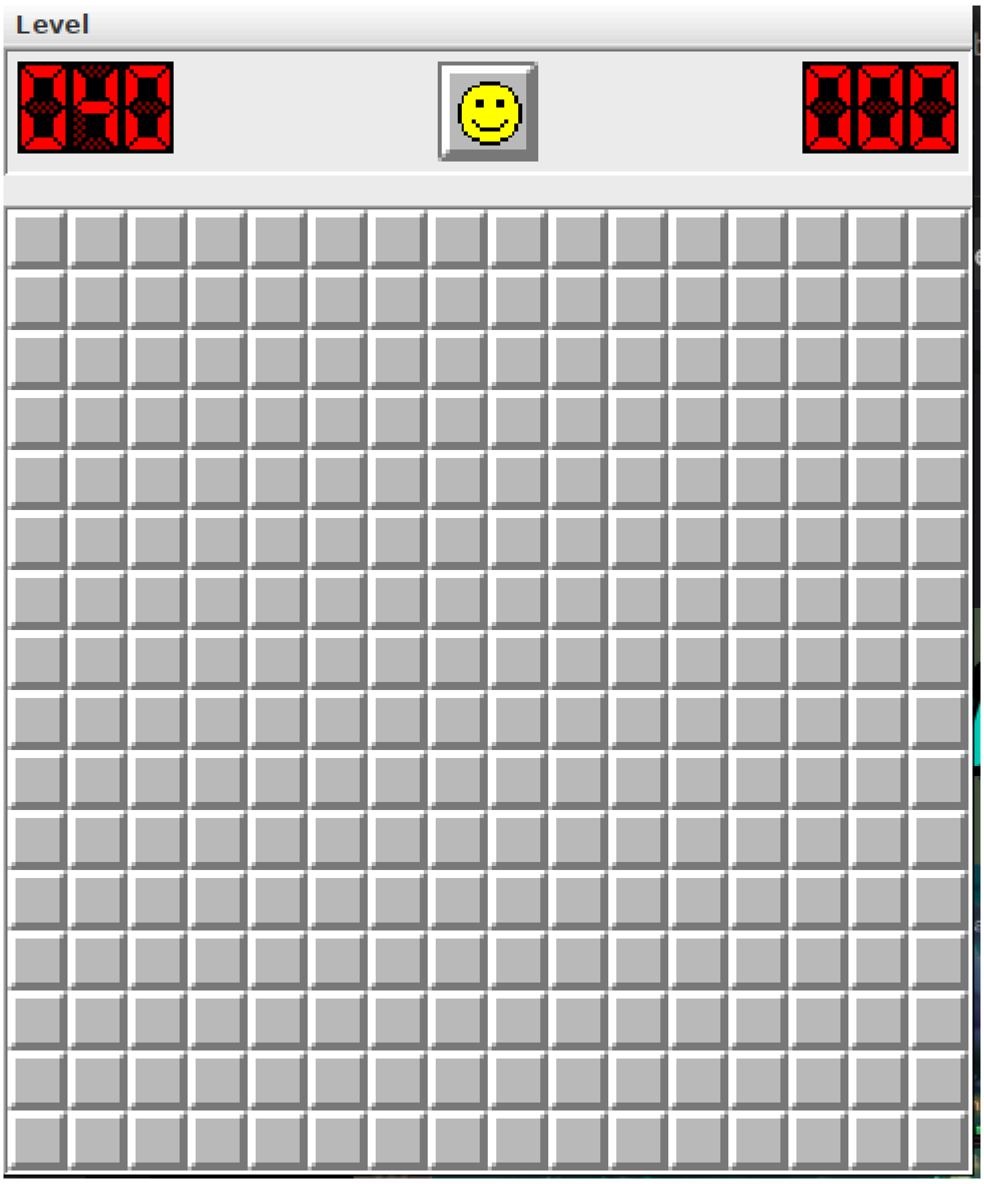


Figure 18. Intermediate Level

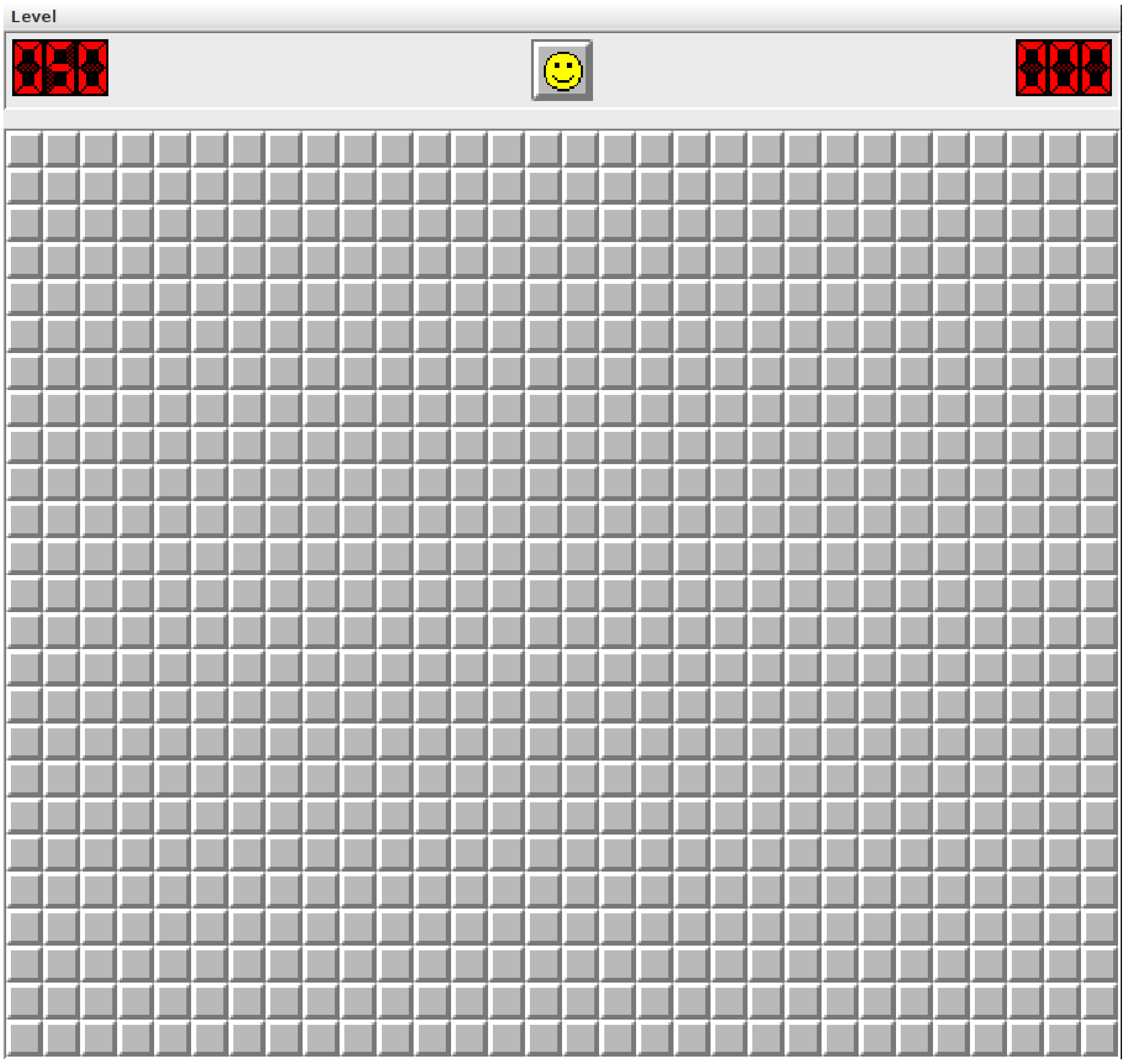


Figure 19. Hard Level

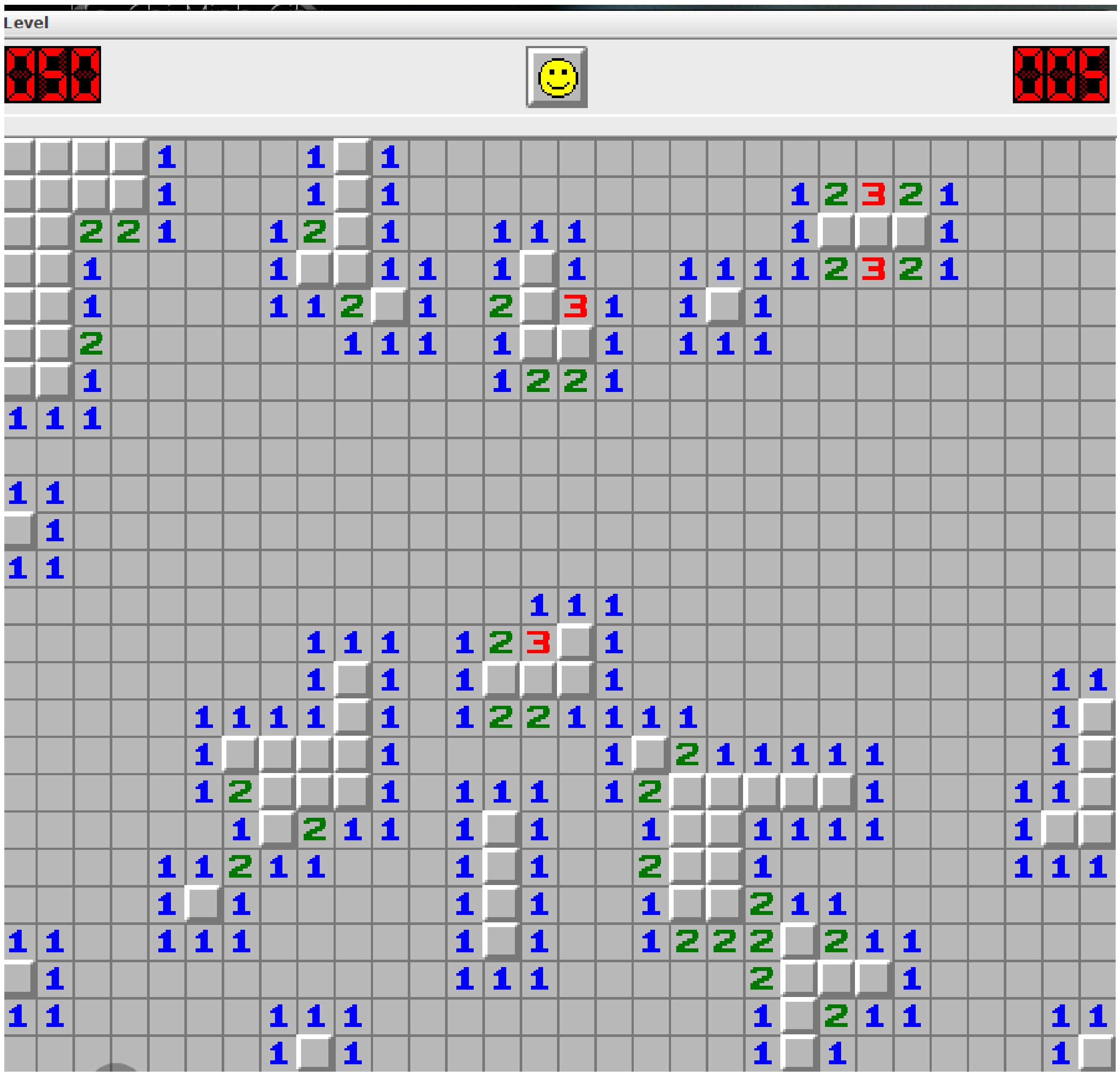


Figure 20, Play the game

CHAPTER 4: CONCLUSION AND FUTURE WORKS

1. Conclusion

The game is currently being developed. I gained a stronger understanding of Data Structure & Algorithm (DSA), the SOLID principles, and the four key Object-Oriented Programming (OOP) components throughout the final term. This game has improved my skills in both OOP and DSA for game development, as well as for programming games that have additional features added after they are completed. The design pattern demonstrates the characteristics of object-oriented programming (OOP), such as inheritance, polymorphous, abstraction, encapsulation, and molecularity. It also demonstrates the application of BFS, recursion, observer and strategy design patterns, and SOLID principles. The wealth of knowledge gained in this endeavour is not only a testament to my collective expertise but also a testament to the innovative spirit that has driven me to push the boundaries of game development. The depth of knowledge I have acquired from this project is evidence of my team's combined experience and my creative energy, which has inspired me to push the limits of game production.

2. Future works

The following are possible careers (future employment) for the Minesweeper game:

1. Expanded the game to enable multi-player mode, enabling players to compete with one another in labyrinth solving or to play against one another.
2. Enhanced User Interface: Designed to deliver a more seamless and captivating gaming encounter. enhanced colours, sound effects, and tile display to boost enjoyment and interaction.
3. Statistics and ranking: Include player score ranking and useful statistics. enables players to assess how well they can calculate against other players, which encourages them to do better.
4. Integrated tip feature: Add a tip feature so players can get help when encountering difficulties in the game. However, careful technique is required to ensure the game's balance and tuning are not affected too much.

--> The work here opens up many opportunities to improve and expand the Minesweeper game, creating a fun and engaging gaming experience for players.

3. Acknowledgement

We would like to convey our deepest appreciation to our instructor and the individuals who assisted us in reaching the goals of this project:

- Dr. Vi Chi Thanh
- Original code from Minesweeper in leetcode

- The sites Geeksforgeeks, Javapoints, JavaSwing and so on
- The README.md template from othneildrew (Drew, 2018/2022)
- Link Code: my source code on github

REFERENCES

*[1]*Minisweeper in leetcode. (n.d) *Retrieved December 11, 2023, from*

https://leetcode.com/problems/minesweeper/description/

https://leetcode.com/problems/minesweeper/submissions/1295488093/

[2] Java Swing Tutorial. Javatpoint. (n.d.). Retrieved December 11, 2023, from

https://www.javatpoint.com/java-swing

[3] UML Diagrams Tutorial. Javatpoint. (n.d.). Retrieved January 4, 2023, from

https://www.javatpoint.com/java-swing

[4] Drew, O. (2022). Othneildrew/Best-README-Template. Retrieved January 4,

2023, from

https://github.com/othneildrew/Best-README-Template

[5] SOLID Principle in Programming: Understand With Real Life Examples.

GeeksforGeeks. Retrieved December 28, 2023, from

https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/

[6] Playing music in a Java game. StackOverflow. Retrieved December 16, 2023,

from

https://stackoverflow.com/questions/27854171/playing-music-in-a-java-game

[7] Data Structure & Algorithm: Understand With Real Life Examples.

. Retrieved December 28, 2023, from

https://techdevguide.withgoogle.com/paths/data-structures-and-algorithms/

[8] Data Structure & Algorithm: Understand With Real Life Examples.

. Retrieved December 28, 2023, from

https://techdevguide.withgoogle.com/paths/data-structures-and-algorithms/