

**Algorithms & Data Structures IT013IU**

**Course Instructor:**

Thai Trung Tin

**Submitted by:**

Do Hoang My Dung – ITITWE21040

**Boom Field**

**FINAL PROJECT**

**The Boom Field** is a classic single-player puzzle game where the objective is to clear a grid of hidden mines without detonating any of them. Each cell may either be empty or contain a mine. The game combines logic, strategy, and a bit of luck to successfully uncover all non-mine cells.

[**CHAPTER 1: INTRODUCTION** **4**](#_Toc23675)

[1.1 Gaming in the Field 4](#_Toc23676)

[1.2 About the Game Project 4](#_Toc23677)

[1.3 Boom Field 5](#_Toc23678)

[1.4 References 5](#_Toc23679)

[1.5 Developer Team 6](#_Toc23680)

[**Chapter 2: SOFTWARE** **6**](#_Toc23681)

[2.1 My project’s result 6](#_Toc23682)

[2.2 My goals 6](#_Toc23683)

[2.3 Use Case Scenario 7](#_Toc23684)

[2.5 Use Case Diagram 7](#_Toc23685)

[2.6 Class Diagram 8](#_Toc23686)

[**Chapter 3: DESIGN AND IMPLEMENTATION** **8**](#_Toc23687)

[3.1 Package Diagram 8](#_Toc23688)

[3.2 File Overview 9](#_Toc23689)

[3.3 Design 9](#_Toc23690)

[3.4 Implementation 10](#_Toc23691)

[a. Core Classes: 10](#_Toc23692)

[b. Key Functionality: 10](#_Toc23693)

[c. Win - Lose Condition 16](#_Toc23694)

[1. Win Condition 16](#_Toc23695)

[2. Lose Condition 16](#_Toc23696)

[Code Handling the Lose Condition 16](#_Toc23697)

[3. Reset Mechanism 17](#_Toc23698)

[4. Tile Revealing Logic 18](#_Toc23699)

[5. Mine Detection 18](#_Toc23700)

[3.5 The complexity 19](#_Toc23701)

[**CHAPTER 4: FINAL APP GAME** **20**](#_Toc23702)

[1. Begin the Game: 20](#_Toc23703)

[2. How to play: 22](#_Toc23704)

[**CHAPTER 5: EXPERIENCE** **23**](#_Toc23705)

# CHAPTER 1: INTRODUCTION

### 1.1 Gaming in the Field

Video game development is a rapidly growing field within software engineering, presenting unique challenges that require a combination of creativity, technical skills and an understanding of user engagement. Game development involves different aspects, such as design, programming, and interaction, all coming together to create an enjoyable and immersive experience for players.

In the **Boom Field** project, I developed a minesweeper-like game as part of the "Data Structures and Algorithms" course, a key component of our degree program. This project allows us to apply theoretical knowledge from class, including **object-oriented programming, 2D arrays, stack** and **search algorithms**. Through this hands-on work, we aim to refine our development skills and enhance our understanding of how to structure code efficiently while building an interactive and enjoyable game.

### 1.2 About the Game Project

There are many similar games like **Minesweeper** available on platforms like Google Play, App Store and web games. However, most of these games operate in the same way: players try to uncover tiles without hitting mines, which can become monotonous as the gameplay repeats itself.

Recognizing this issue, our team has revamped the classic Minesweeper game by adding a modern interface and new features. In **Boom Field**, i enhanced the game with different color schemes, providing a fresh and engaging experience for players. The game board is designed with vibrant colors, making it easier for players to distinguish between tiles and create a more enjoyable atmosphere while playing.

In short, our goal is to offer a more appealing and engaging gaming experience for users, encouraging long-term play and building player loyalty through improved interface design and fun gameplay in **Boom Field**.

### 1.3 Our “Boom Field”

The **Boom Field** game offers simple but engaging gameplay, where players navigate a grid of tiles to avoid hidden mines and clear the board. To enhance user experience, we’ve implemented intuitive features that make the game enjoyable and accessible for players of all skill levels.

The game includes the following structure and features:

* **Main Menu**: A clear and minimalistic interface with a **Play** button to start the game.
* **Difficulty Levels**: Players can choose from three levels:
  1. **Easy**: Contains 10 mines.

○ **Medium**: Contains 20 mines.

○ **Hard**: Contains 30 mines.

* **Gameplay Flow**: After finishing a game, players can either:
  1. **Play Again**: Restart with the same difficulty level.

○ **Back to Home**: Return to the main menu to select a different level or quit the game.

* **Customization**: Players can select different color themes to personalize their experience.

The **Boom Field** project highlights the application of **object-oriented programming (OOP)** principles and the use of fundamental **data structures and algorithms** to manage gameplay elements like the grid, mine placement, and user interactions.

By incorporating these features, **Boom Field** offers a polished and enjoyable gaming experience while allowing us to apply and deepen our understanding of OOP concepts in a real-world project.

### 1.4 References

Minesweeper: <https://github.com/ImKennyYip/minesweeper-java/blob/master/Minesweeper.java><https://www.youtube.com/watch?v=5VrMVSDjeso>

“Play Again” Button: <https://www.youtube.com/watch?v=cA1GvZ5Y3-U>“Play” Button: <https://youtu.be/aOcow70vqb4>

### 1.5 Developer Team

**Boom Field** was created by **Đỗ Hoàng Mỹ Dung,** a Computer Science students from International University.

I did: Research, Coding, Write report, Make slides

# Chapter 2: SOFTWARE

### 2.1 My project’s results

1. A user-friendly, efficient and engaging gameplay experience.
2. Minimal maintenance cost, simple and funny graphics.
3. Compatibility with expected PC/mobile configurations, with plans to develop using Android Studio in the future.
4. Intuitive and easy-to-use interface for all ages.
5. Professional and optimized coding for high performance and reliability.

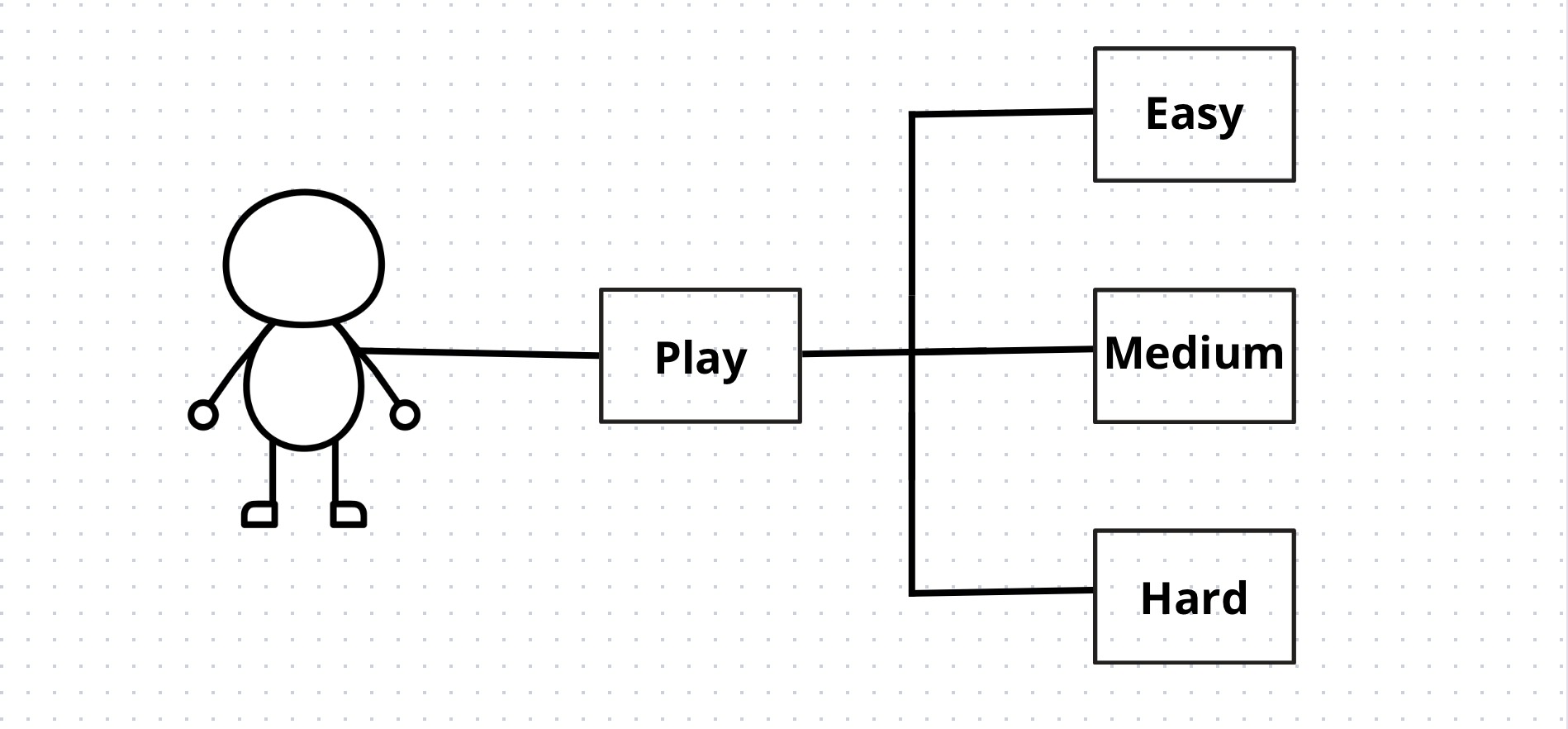
### 2.2 My goals

1. A simple, childhood-inspired game suitable for all ages.
2. Cute and visually appealing colors that attract players.
3. Focus on developing intellectual skills through gameplay.
4. An offline game that doesn’t require constant internet access.
5. Designed to provide a relaxing and enjoyable experience for everyone.

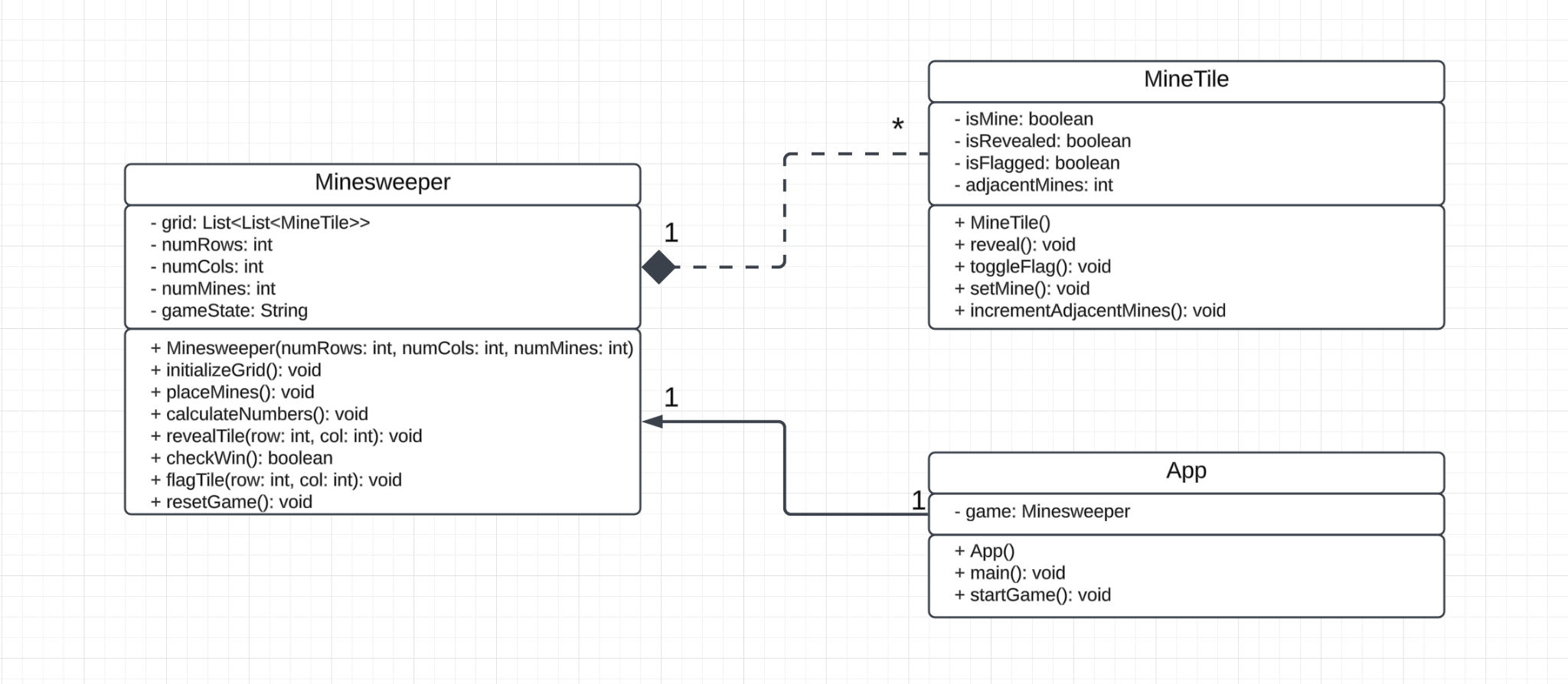
### 2.3 Use Case Scenario

|  |  |  |
| --- | --- | --- |
| Boom Field | Play | Select a level. |
| Easy | Include 10 boom traps. |
| Medium | Include 20 boom traps. |
| Hard | Include 30 boom traps. |
| Play Again | Replay. |
| Back Home | Return to the main screen to reselect the level. |

### 2.5 Use Case Diagram



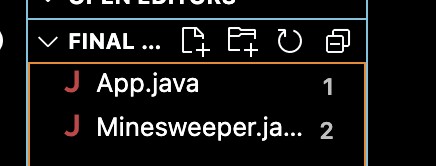
### 2.6 Class Diagram



# Chapter 3: DESIGN AND IMPLEMENTATION

### 3.1 Package Diagram

**Packet Diagram** visually represent how different components or classes in a program interact with each other. In this project, there are two primary Java files, and the diagram illustrates the relationship between these files and how they are organized within the structure of the project. The diagram provides a clear overview of how the game logic and user interface components work together to create the Minesweeper game.



### 3.2 File Overview

**App.java:** this file server as the entry point for the Minesweeper application. The responsibility of this is initializing the game and setting up the GUI components. The **app.java** file contains the main() method, which starts the program and integrates the user interface with the game logic.

* **Game Initialization:** set up the grid and begin a new game by calling methods form Minesweeper.java
* **User Interface Setup:** the GUI components including buttons for grid and other UI elements are created by User Interface Setup. This ensures that the interface is responsive to player interactions.
* **Event Handling:** it solves the input information or requirements of players. It also communicates with the minesweeper.java to update the game state.

**Minesweeper.java**: this file contains the core game logic. It defines the rules of Minesweeper, including mine placement, uncovering cells, and checking for win or loss conditions. It does not interact directly with the user interface but provides methods to operate and track the game state.

* **Grid Setup:** defines a 2D array to represent the game grid, where each cell can either contain mine or a number (indicating how many adjacent mines is has)
* **Mine Placement**: randomly places mines in the grid while ensuring that there are no conflicts with adjacent cells.
* **Stata Tracking**: manages the game’s state, including the number of uncovered cells, flagged cells, and checking for win/loss conditions.

### 3.3 Design

This report details the design and implementation of a graphical java application replicating the classic logic puzzle game “Minesweeper” with a design layout “Boom” theme, title “Boom Field”. The application adheres to established game mechanics while incorporating a unique aesthetics and user interface.

* **User Interface (UI):**
  + The UI utilizes the Java Swing library for component creation and layout management
  + A **JFrame** serves as the main window, titled “Boom Field”
  + A **JPannel** displays the game board, employing a GridLayout for a grid-like structure.
  + Individual cells of the board are represented by custom **MineTile** objects extending **JButton**.
  + A separate **JPanel** displays the current game state and mine count using a **JLabel**.
  + A final **JPanel** houses the "Play Again" and "Back Home" buttons for game control.
* **Gameplay Mechanics:**
  + The game adheres to the standard Minesweeper rules:
  + Players uncover hidden cells, revealing numbers indicating adjacent mines or remaining safe spaces.
  + Clicking a mine cell triggers a game-over state, revealing all mine locations.
  + Right-clicking a cell allows players to flag potential mines.
  + The "Boom Field" theme is integrated through:
  + A gray boom symbol ("💣") represents a mine.
  + A dark-red boom symbol ("💣") is used for user-placed flags.

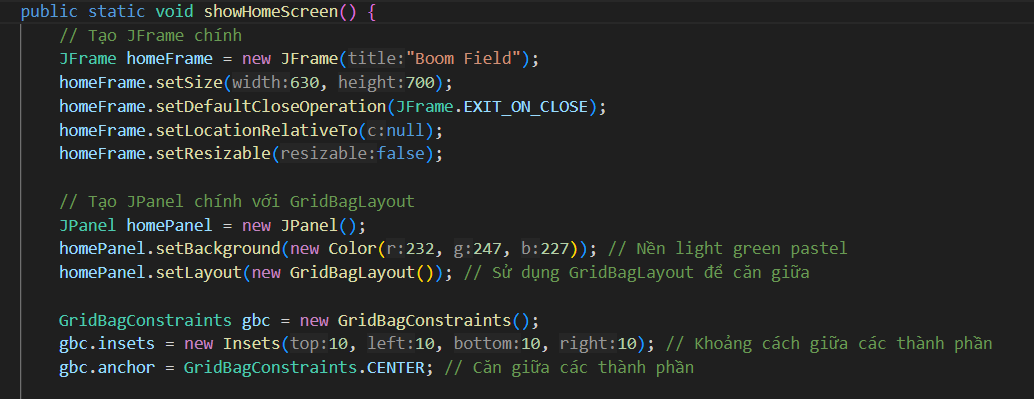
### 3.4 Implementation

#### a. Core Classes:

* App: Handles the home screen display and initiates the Minesweeper game.
* Minesweeper: Manages the game logic, board initialization, and user interaction.
* MineTile: Extends JButton to represent individual cells on the game board, handling mouse events and cell state updates.

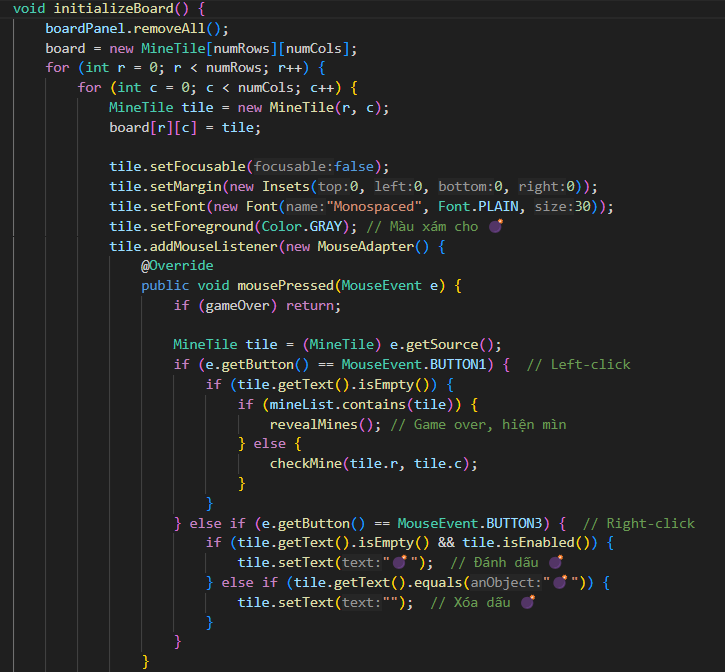
#### b. Key Functionality:

- showHomeScreen: Creates the home screen layout with a welcome message, instructions, and a "Play Game" button.



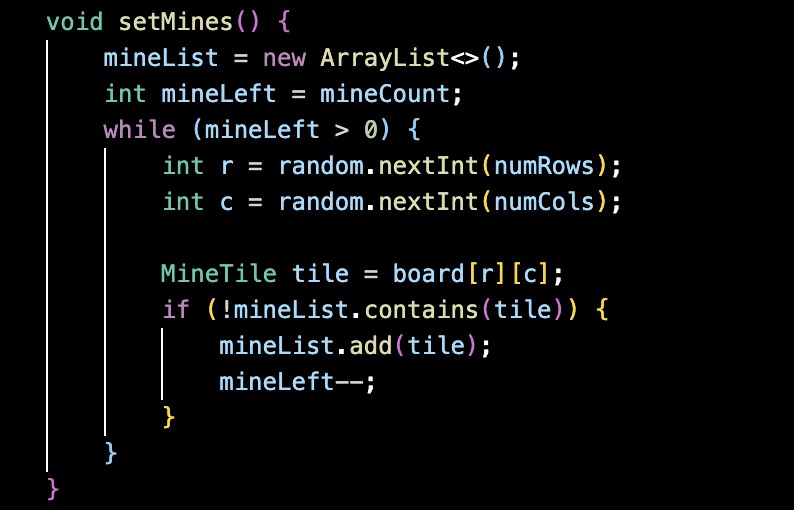
**How it works:**

* showHomeScreen function creates and displays the home screen of the "Boom Field" game
* JFrame (main window) contains a JPanel (homePanel) that is organized using GridBagLayout
* GridBagLayout organizes UI elements like a boom symbol, welcome message, instructions, and a "Play now" button, all centered on the screen
* These elements are styled and spaced using GridBagConstraints for precise positioning
* initializeBoard: Creates MineTile objects for each cell, setting visual properties and mouse listeners.



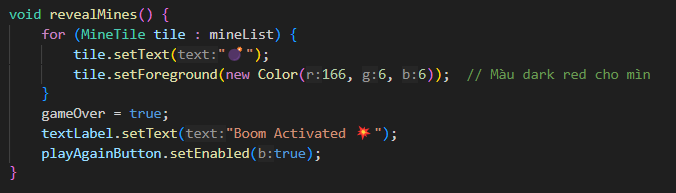
**How it works:**

* + MineTile[][] board: Creates a 2D array of MineTile objects representing each cell on the game board.
  + Every row and column, a MineTile object is instantiated and stored in the board array.
  + Left-click checks if the tile contains a mine. If it does, the game ends by calling revealMines(). If not, checkMine(r, c) continues gameplay.
  + Right-click toggles a boom marker (💣) on or off the tile to mark a suspected mine.
* setMines: Randomly distributes a defined number of mines across the game board.



**How it works:**

* + The mineList is cleared to reset any previously placed mines.
  + Random row (r) and column (c) indices are generated using Random.
  + Mines are added to the mineList only if the tile at (r, c) is not already in the list.
  + The process repeats until the total number of mines equals the defined mineCount.
* revealMines: Uncovers all mine locations and displays a game-over message.



**How it works:**

* + Iterates through the mineList to uncover all mine tiles by setting their text to a bomb icon.
  + Disables further interaction with mine tiles to prevent additional user actions.
  + Shows a message dialog using JOptionPane, informing the player of the game-over event.
  + Sets the gameOver flag to true, halting further gameplay.
* checkMine: Handles left-click events on a cell, revealing adjacent mine count or initiating recursive exploration for safe areas.

A screen shot of a computer program

AI-generated content may be incorrect.

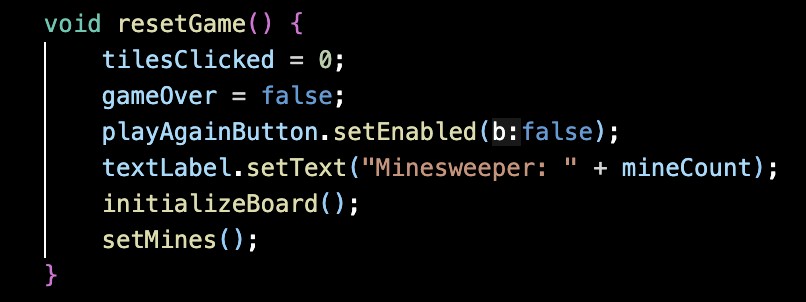
**How it works:**

* + It first checks whether the coordinates (r, c) are within the board bounds to avoid out-of-bounds errors.
  + The code uses a stack to keep track of the tiles that need to be checked. It pushes the starting tile into the stack.
  + Title Processing iteratively pops tiles from the stack. Skips already revealed tiles and marks the tile as revealed and increments the count of clicked tile.
  + The process continues, revealing adjacent safe tiles, until all connected empty tiles are uncovered.
* countMine: Checks if a neighboring cell contains a mine.



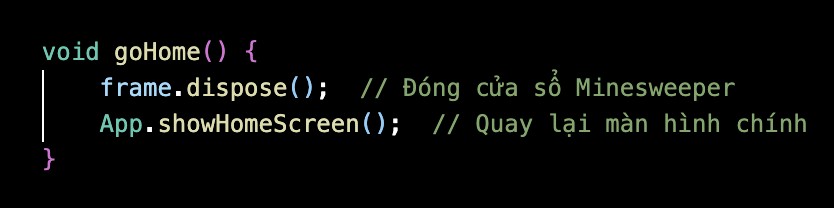
**How it works:**

* + Checks all adjacent tiles (8 neighbors) to see if they contain mines.
  + Counts how many of these neighboring tiles are mines, returning that number.
  + Determining the safe areas and how many mines surround a particular tile, which is displayed on the tile.
* resetGame: Resets the game state, clearing the board and restarting the game.



**How it works:**

* + Method is responsible for resetting the state of the game.
  + Removes any markings and reveals all previously hidden tiles, effectively resetting the board.
  + Reinitializes the game with the original number of rows, columns, and mines, and sets the game state to "active" again.
* goHome: Closes the Minesweeper window and returns to the home screen.



**How it works:**

* + Disposes of the current Minesweeper window, closing the game.
  + Triggers the display of the home screen, allowing the user to navigate back to the main menu.

#### c. Win - Lose Condition

##### 1. Win Condition

* The player wins by clicking all non-mine tiles.
* This is determined by checking if the number of tiles clicked equals the total number of non-mine tiles.

**Code Handling the Win Condition**

* The win condition is implemented in the checkMine() method, specifically in this part:

A screen shot of a computer code

AI-generated content may be incorrect.

**How it works:**

* Each time a tile is revealed, the tilesClicked counter increments.
* When the number of revealed tiles matches the number of non-mine tiles, the game sets gameOver to true, updates the textLabel to indicate victory, and enables the "Play Again" button.

##### 2. Lose Condition

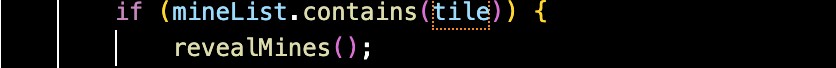
- The player loses if they click on a tile containing a mine. When this happens:

+ All mines are revealed.

+ The game ends, and the player cannot continue interacting with the board.

##### **Code Handling the Lose Condition**

- The lose condition is handled in the mousePressed() method within the MouseAdapter for each tile. Specifically:



When a mine is clicked:

1. The revealMines() method is called.
2. The following code reveals all mines and ends the game:

A computer screen shot of text

AI-generated content may be incorrect.

**How it works:**

* Each mine in the mineList is visually revealed by setting its text to "💣" (the boom symbol) and coloring it dark red.
* The gameOver flag is set to true, stopping further interaction with the board.
* The textLabel displays a lose message, and the "Play Again" button is enabled.

##### 3. Reset Mechanism

- For both win and lose scenarios, the "Play Again" button allows the player to restart the game. This is handled by the resetGame() method:



**How it works:**

● The number of clicked tiles and the game state are reset. ● A new board is generated with fresh mines.

##### 4. Tile Revealing Logic

* The checkMine() method handles revealing tiles recursively. If the clicked tile has no adjacent mines, it reveals the surrounding tiles automatically:



* If the tile has adjacent mines, it displays the number of nearby mines.
* If no adjacent mines are found, the surrounding tiles are revealed recursively using a stack.

##### 5. Mine Detection

* The number of adjacent mines is calculated using the countMine() method:



* Check each surrounding tile.
* Returns the count of mines in adjacent tiles.

### 3.5 The complexity

* **checkMine**:
  + **Best Case**: O(1) -> When the clicked tile is already checked or has no adjacent empty tiles.
  + **Worst Case**: O(n x m). -> When all tiles are connected and safe, the entire board is explored. n is the number of rows, m is the number of columns.
* **revealMines:**
  + **Best Case:** O(k), where k is the number of mines (each mine is placed in one operation without collision
  + **Worse Case:** O(k ⋅ n ⋅ m) -> In case of repeated collisions, where n x m is the board size.
* **initializeBoard:**
  + **Best Case and Worst Case:** O(n×m) -> Every cell in the n×mn grid is initialized and configured.
* **setMines:**
  + **Best Case**: O(k), where k is the number of mines. -> No collisions while placing mines.
  + **Worst Case**: O(k⋅n⋅m) -> If collisions frequently occur, requiring repeated attempts to place mines on the n×m board.
* **countMines:**
  + **Best and Worst Case**: O(8) = O(1), as it checks a fixed maximum of 8 neighbors, regardless of board size.
* **resetGame:**
  + **Best and Worst Case**: O(n x m), as it clears and reinitializes all tiles.
* **goHome:**
  + **Best and Worst Case**: O(1), as it involves a simple window management operation.

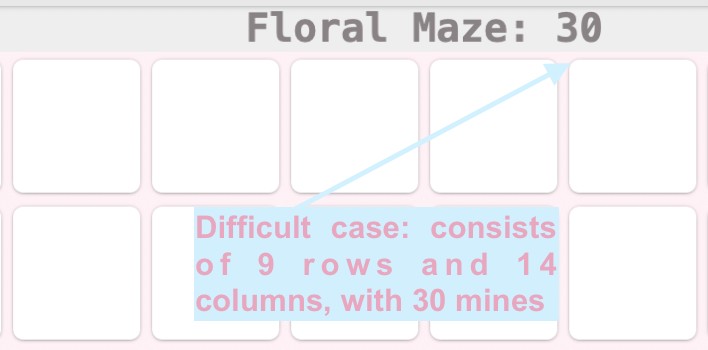
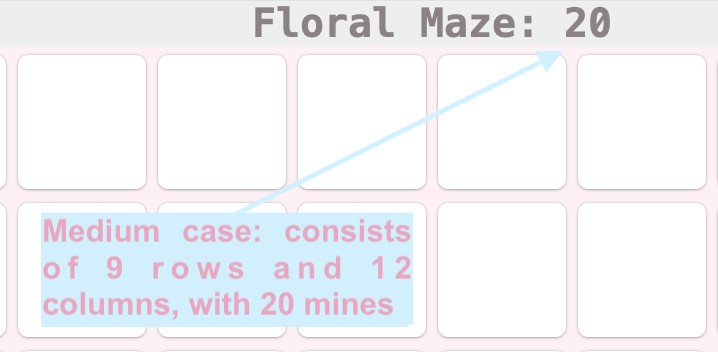
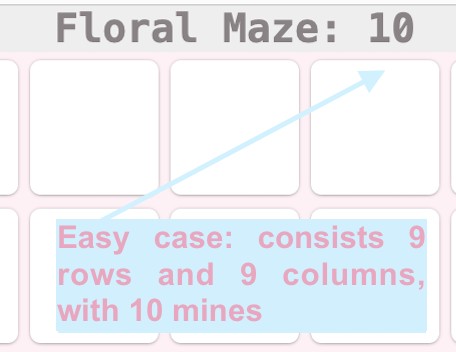
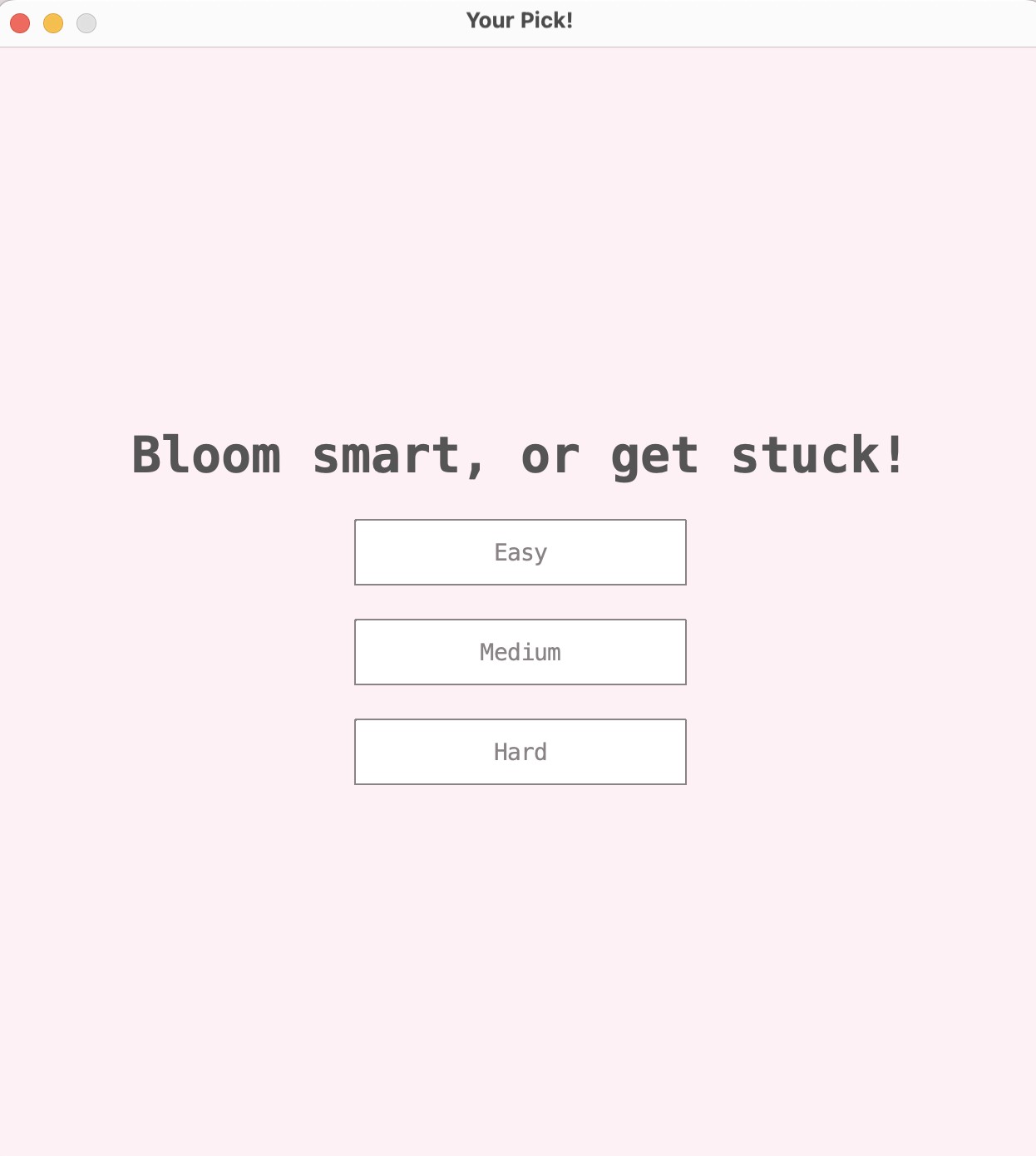
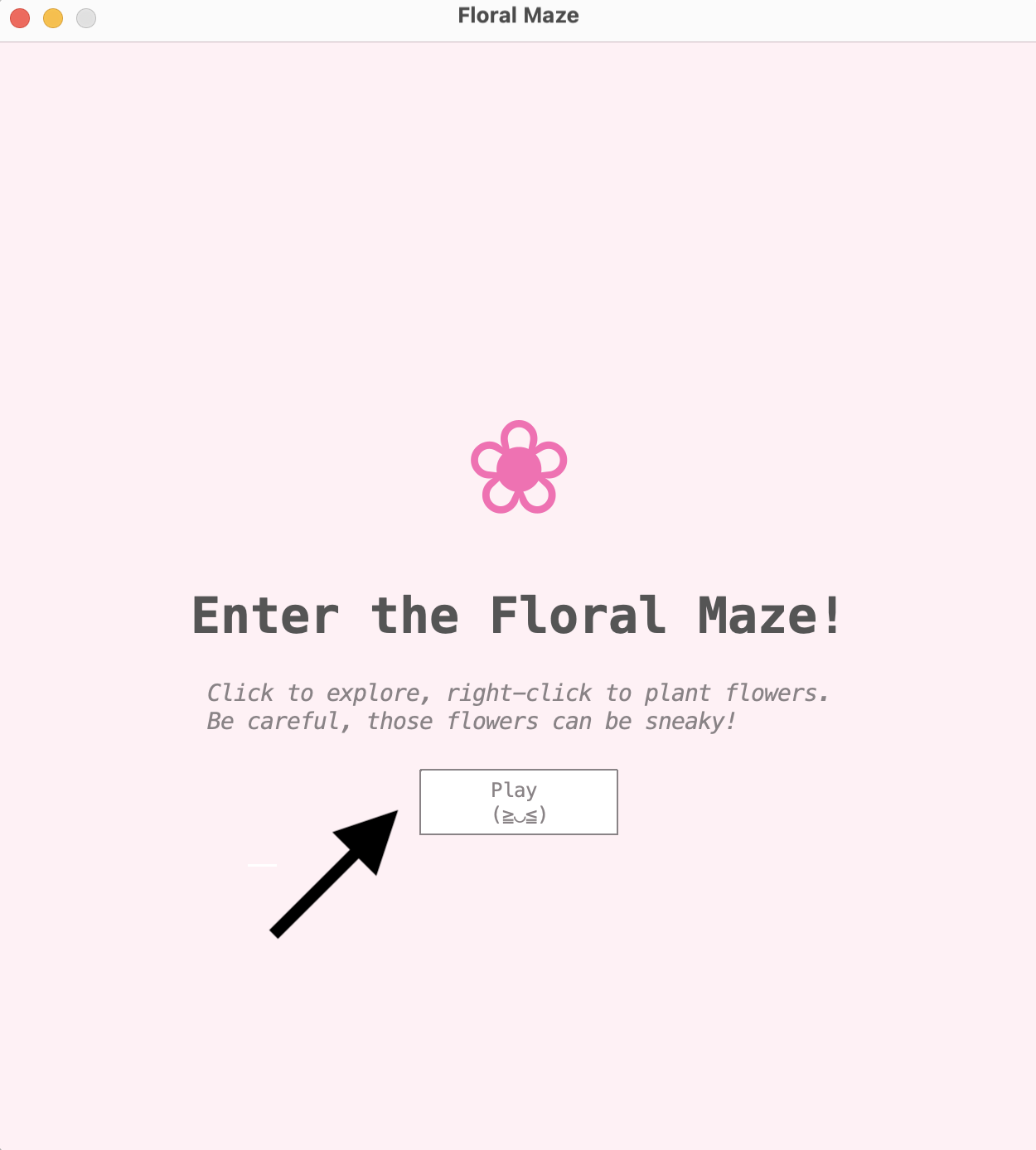
# CHAPTER 4: FINAL APP GAME

## 1. Begin the Game:

Click on the button “play” on the Home screen of “Boom Field” to start a new game.

When the player clicks the "Play" button, a new interface appears, allowing them to select from three main difficulty levels: Easy, Medium, and Hard.

Then, after the player finishes choosing the levels, the main screen will appear to start.

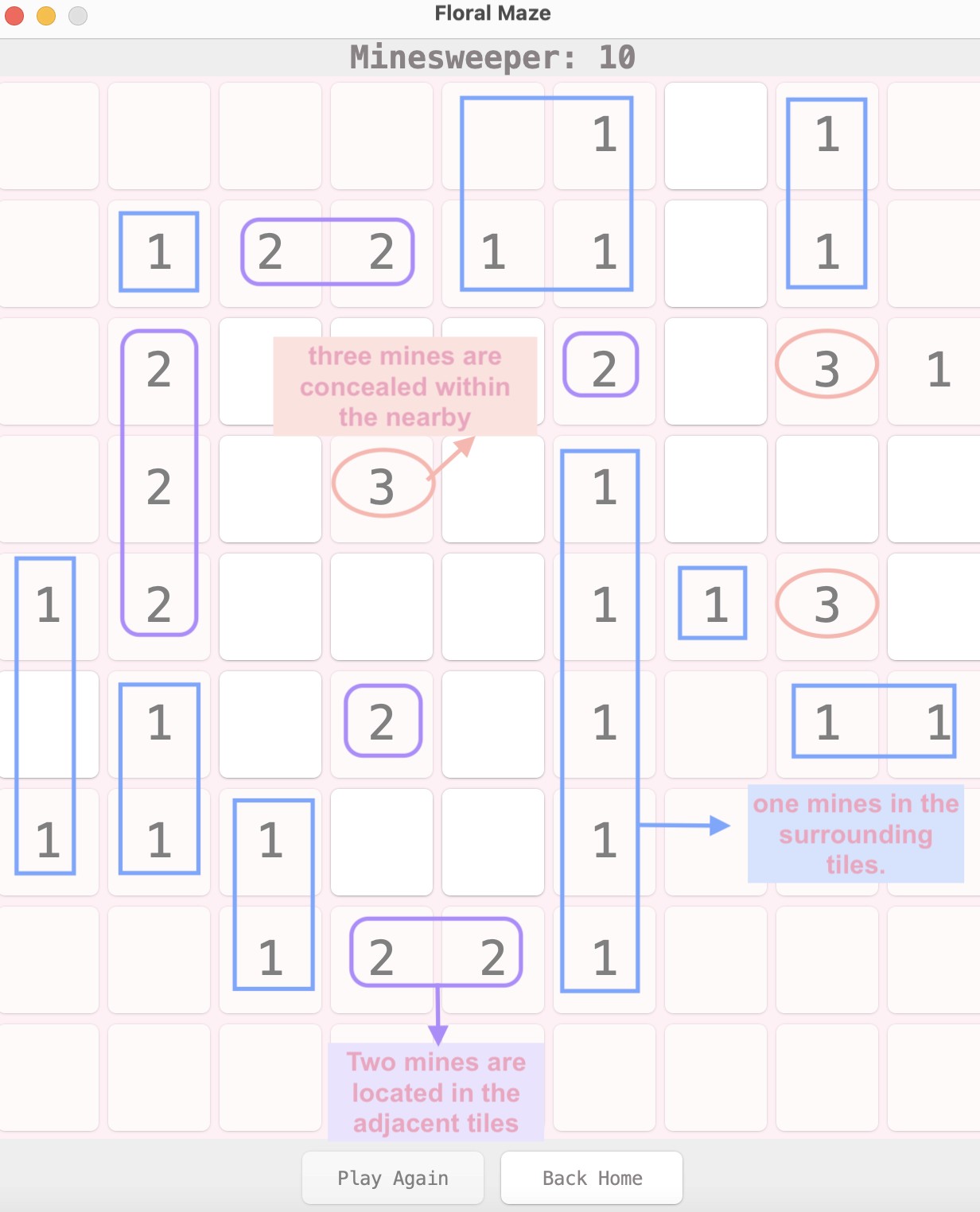


## 2. How to play:

Click any title to begin, so when a player clicks a safe title, one of three possible numerical values (1, 2 or 3) will be displayed.

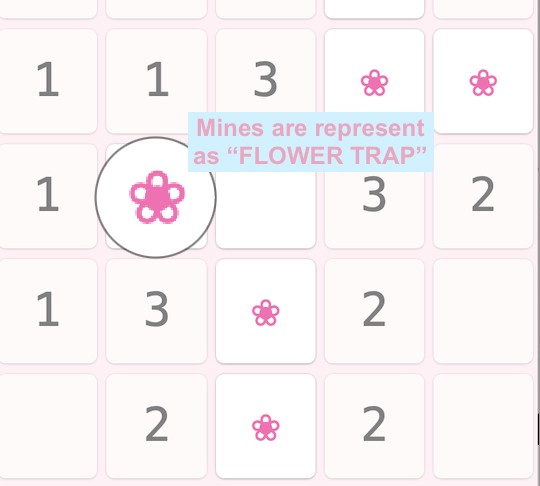
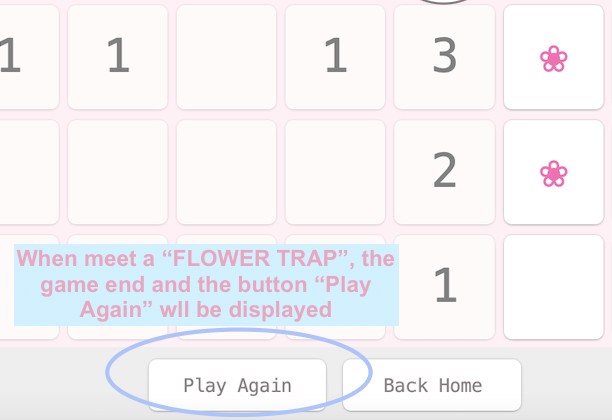
+ 1: There is one mine in the surrounding tiles. + 2: Two mines are located in the adjacent tiles + 3: Three mines are concealed within the nearby.

=> These numerical values serve as logical clues for players to deduce the locations of hidden mines and strategically plan their next moves.



If a player clicks on a title that contains a hidden mine, which was called a boom trap, the game immediately ends. This is visually represented by revealing the mine and exposing all remaining mines on the grid.

The button “Play Again” will be activated, and the player will click it to play again.



# CHAPTER 5: EXPERIENCE

Creating **Boom Field** has been an exciting and eye-opening experience for our team. We’ve realized that **making a game is about so much more than just writing code**. A great game needs engaging gameplay, a rich story, stunning artwork, and smooth animations to truly come to life.

Through this project, we’ve applied what we’ve learned in class while tackling bugs and solving unexpected challenges. It’s been a chance to strengthen our knowledge, explore new technologies, and gain valuable hands-on experience.

We also learned that being a Computer Science student means embracing self-study. The IT world is vast, and curiosity is the key to keeping up with it.

**Boom Field** is just the beginning. We’re excited to share this project with you soon and aim to create more games and apps that bring joy and value to users.

Stay tuned—**Boom Field** is coming your way!