

# **Basics of Programming 3**

## **Homework Project**

**Name:** Muhammad Hameez Khan

**Neptun:** TFBB32

**Project Title:** Bomber Man Game

**Homework Documentations and details**

**Bomberman Game Project Documentation**

## Description and Overview

The **Bomberman Game Project** is a Java-based desktop application inspired by the classic Bomberman game. This project allows players to navigate a grid-based game board, place bombs, and destroy enemies while avoiding obstacles and managing limited lives. The project integrates several functionalities to enhance gameplay and user experience.

### Core Features:

- **Game Setup:** Initializes a 13x13 grid-based board with walls, breakable tiles, a player, and multiple enemies.
- **Player Movement:** Navigate using keyboard controls (W, A, S, D) while interacting with the environment.
- **Bomb Placement and Explosion:**
  - Players can place up to three bombs simultaneously.
  - Bombs have a countdown timer and explode in four directions, affecting enemies and walls.

- **Enemy AI:** Enemies move autonomously in predefined patterns, avoiding obstacles and pursuing the player.
- **Game State Management:**
  - Save the current game state, including the player, enemies, and active bombs, using Java serialization.
  - Load saved games to resume progress seamlessly.
- **Graphical User Interface (GUI):** Provides an intuitive interface for gameplay and navigation through menus.
- **Game Over and Victory Conditions:**
  - The game ends if the player loses all lives.
  - Victory is achieved by destroying all enemies.

### **Technologies and JAVA Techniques Used:**

- **Java Swing:** For creating the graphical user interface. It provides the framework for components such as panels, buttons, menus, and windows.
- **JUnit 5:** For testing functionalities, validating code behaviour and ensuring code reliability.

- **Graphics Class:**

The Graphics class in Java is crucial for rendering game elements on the screen. It is used within the GameBoardPanel class to draw the grid layout of the game. Specific colors are assigned to each game element (e.g., black for walls, blue for players) to visually distinguish them. Additionally, the Graphics class handles real-time rendering of the game state.

- **JTable:**

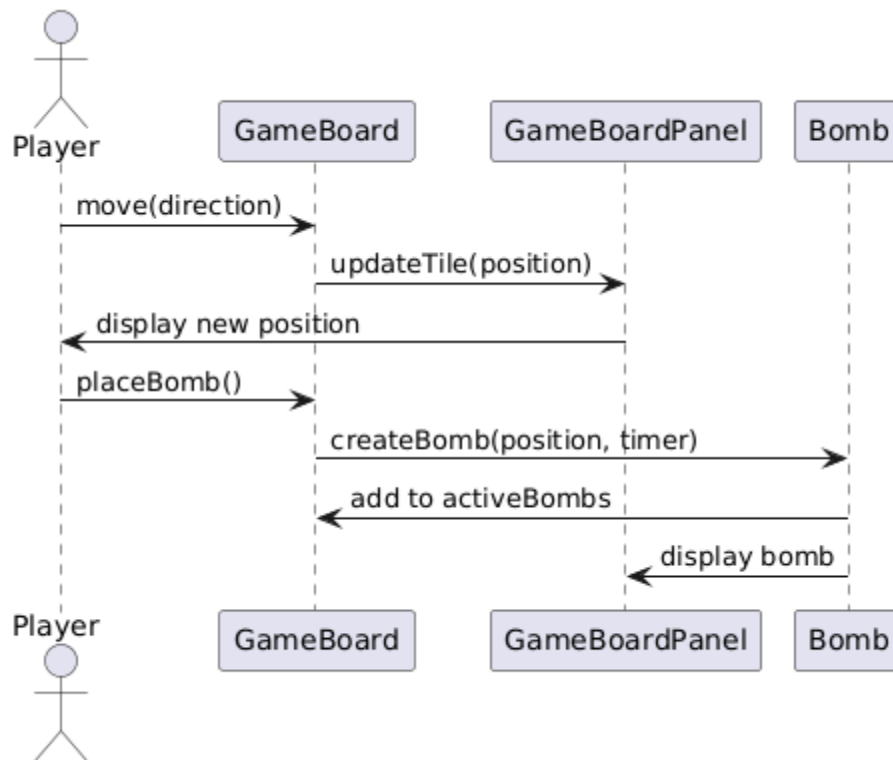
The JTable class is used in the game's main menu to present options to the player. The table displays three choices "Start New Game," "Load Saved Game," and "Exit Game." The rows in the JTable allow the player to select and activate options through mouse clicks. The JTable's appearance is customized with fonts and row heights to enhance readability and aesthetic appeal.

## **2. Sequence Diagrams**

**The following sequence help to better understand the mechanics of the program:**

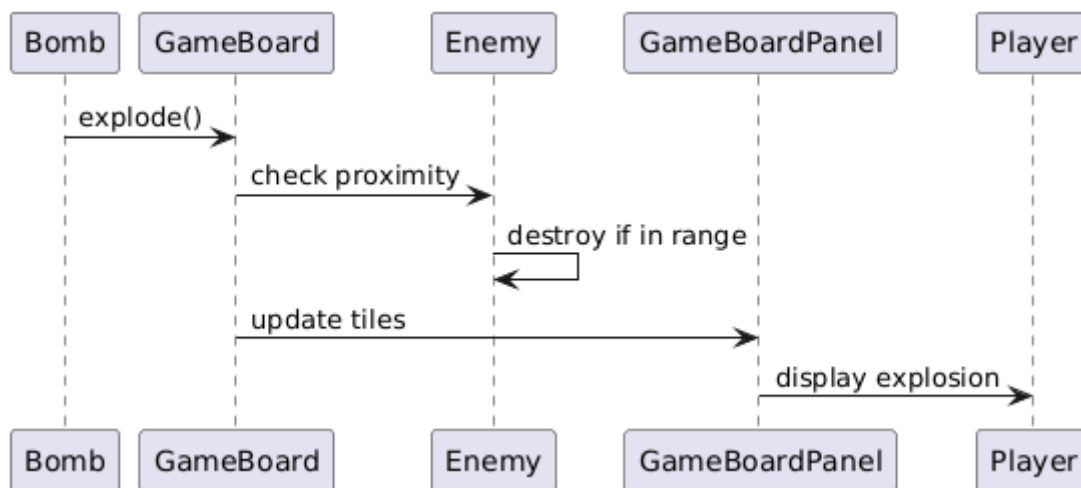
## 2.1: Player Moves and Places a Bomb

The player moves, updates their position on the board, and places a bomb if allowed.



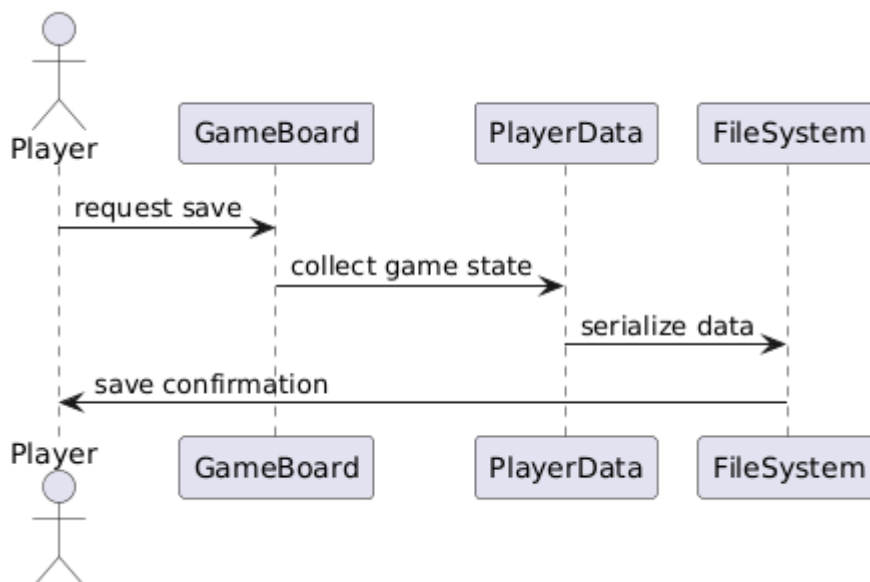
## 2.2: Bomb Explodes and Affects Enemies

A bomb explodes, destroys nearby enemies, and updates the board.



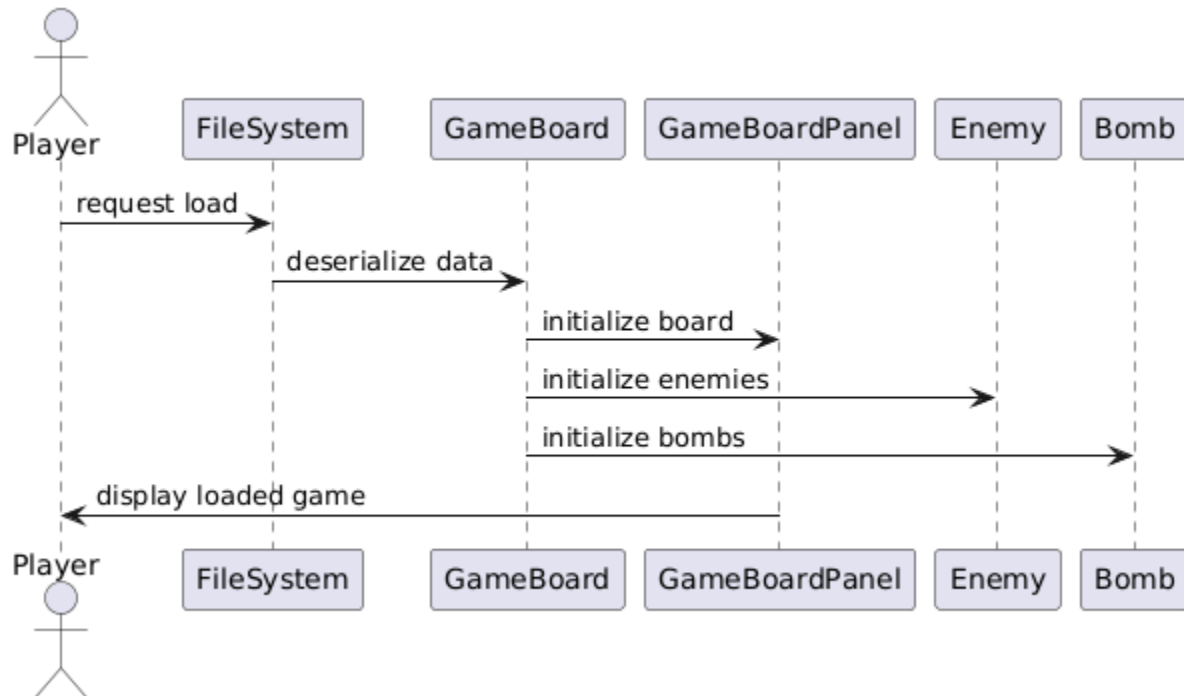
## 2.3: Save Game and Exit

The current game state is serialized and saved to a file.



## 2.4: Load Saved Game

A saved game state is deserialized and restored to resume gameplay.



## 3. Key Functionalities

### 3.1 Player Actions:

- **Movement:** Controlled using the W (up), A (left), S (down), and D (right) keys.
- **Bomb Placement:** Spacebar places a bomb at the player's current location. Explosions can destroy breakable walls and enemies. The explosions can also take players life.

### 3.2 Enemy Behavior:

- Enemies patrol the board in predefined directions, switching directions when encountering obstacles.
- Enemies damage the player upon collision.

### **3.3 Bomb Mechanics:**

- Bombs explode after a 3-second timer, clearing tiles in 3 radius diameter.
- Breakable walls are destroyed, while indestructible walls block the explosion.
- Bomb explosions damage enemies and the player if caught in the blast radius.

### **3.4 Game Board:**

- A 13x13 grid with:
  - **W**: Indestructible walls.
  - **B**: Breakable walls.
  - **E**: Empty spaces.
  - **P**: Player.
  - **A**: Enemies.
  - **G**: Bombs.

### **3.5 Save and Load:**

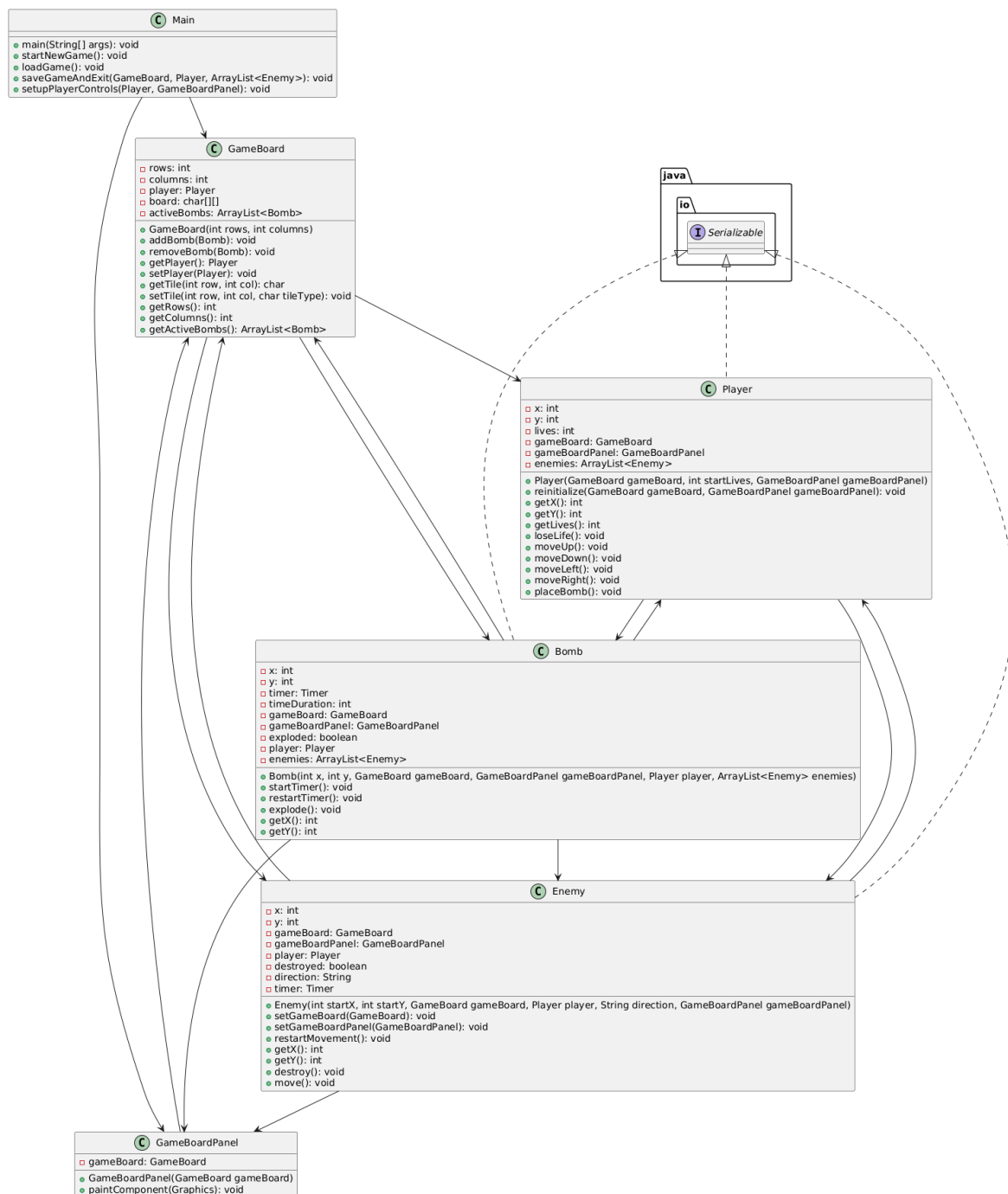


- Save the current game state, including the player's position, enemies, and active bombs, to a serialized file.
- Load saved games to restore progress.

### **3.6 Game States:**

- **Victory:** Achieved by destroying all enemies.
- **Game Over:** Triggered when the player loses all lives.

## **4. Class Diagram:**



## 5. Classes Implemented:

The following classes and their functionalities are part of the system:

### **GameBoard:**

- Represents the game grid. It is basically the map of the game.
- Stores and manages tiles, bombs, and player/enemy positions.

### **GameBoardPanel:**

- Renders the game board on the GUI using Swing.

### **Player:**

- Manages player attributes like position, lives, and bomb placement. Generates bombs.

### **Bomb:**

- Handles bomb countdowns and explosions, affecting the game board.

### **Enemy:**

- Manages enemy movement and interaction with the player.

### **Main:**

- Entry point for the application, managing menus and game state transitions.

---

## **6. Use-Case Scenarios**

## **6.1 Starting a New Game:**

1. The player selects "Start New Game" from the main menu.
2. The game initializes a 13x13 board with walls, the player, and enemies.
3. The player begins navigating the board.

## **6.2 Loading a saved Game:**

1. Loading a saved game is possible with the help of serialization.
2. The program saves the game by serializing it.
3. When a game load is requested the program deserializes the stored objects.

## **6.3 Exit Game:**

1. Upon completing the game the program can be exited with the help of exit game button.

## **6.4 Placing a Bomb:**

1. The player presses the spacebar to place a bomb.
2. The bomb appears on the board and starts a 3-second countdown.

### 3. Upon explosion:

- Tiles in four directions are cleared.
- Breakable walls are destroyed.
- Enemies and the player (if within the radius) are affected.

## **6.5 Saving and Loading a Game:**

1. The player selects "Save Game and Exit" from the in-game menu.
2. The current game state is serialized and saved.
3. On re-launching, the player selects "Load Game" to restore the saved state.

## **7. Unit Testing**

### **Tested Features:**

#### **1. GameBoard:**

- Ensure proper initialization and tile management.
- Validate bomb addition and removal.

#### **2. Player:**

- Test movement constraints and interactions with bombs and enemies.

- Ensure proper bomb placement logic.

### 3. Bomb:

- Validate timer functionality and explosion mechanics.
- Ensure correct handling of breakable walls, enemies, and player damage.

### 4. Enemy:







- Test movement patterns and interactions with obstacles.

### 5. Serialization:

- Validate saving and loading game states, ensuring data integrity.

## Tools Used: JUnit 5

An Adequate test coverage result was achieved. |  
except for the main class whose is to simply setup  
the GUI and initialize objects.

Element	Coverage	covered Instructions	missed Instructions	Total Instructions
▼  bomberman	 67.5 %	2,604	1,253	3,857
>  src	 54.4 %	1,466	1,230	2,696
>  test	 98.0 %	1,138	23	1,161

## 8. Conclusion

The **Bombberman Game Project** successfully implements a fun and challenging gameplay experience. By leveraging object-oriented

programming principles and Java Swing for GUI design, the game achieves both functionality and basic aesthetics.

The project showcases efficient handling of core gameplay mechanics, such as player movement, bomb placement, and explosions, while also incorporating dynamic elements like enemy behavior and breakable walls. The save-and-load functionality enhances the usability of the game by allowing players to pause and resume their progress at any time, to enhance the gaming experience. Testing through JUnit ensures code reliability, making the application robust and maintainable. Robustness and reliability are ensured through extensive testing using JUnit, which validates all major components of the game, including the player, enemies, bombs, and game board.