**COEN 275: Object-Oriented Analysis & Design**

**Professor Leyna Cotran**

**Spring 2018**

**BOMB-FIGHTER**

**Team 7**

**Senbo Yan**

**Zora**

**Sushruth**

**Wei**

1. **GAME OVERVIEW**

Bomb Fighter is a two-player strategy-oriented, action-adventure game where the player has to defeat the opponent by collecting the maximum possible treasures. The original idea of this game comes from a combination of Tank Fighter and Pac-Man games. The treasures are hidden from the view at the beginning of the game and the player has to place the bomb to remove the obstacle and hunt the treasure. The game has a time limit and the player with the maximum number of treasures wins the game. The player may also win the game by bombing the opponent.

The players have to be careful as there may be hidden landmines that may explode on exposure. This makes the game more interesting as the players always have to be alert after placing bombs. The bomb can kill the player that planted it if the player is in the vicinity of the explosion, in which case the opponent wins.



Figure 1. Game menu



Figure 2. Sample Game Screenshot

1. **REQUIREMENTS**
   1. Glossary of Terms

Some terms used in everyday language were redefined within the context of this application along with some unique terms introduced for the first time.

**Bomb-Fighter**: each player is referred to here as the bomb-fighter. There are two characters that a player can choose from, **panda** and **tiger**.

**Bomb**: item placed by the player to destroy the wall so that the player can collect the coins.

**Coin**: the collectible item that either of the players can collect to increase their score, which determines whether they win or not.

**Lollipop**: this is a collectible that increases the Bomb’s power so that a wider area is destroyed.

**Mine**: the bombs that are hidden in the walls along with the coins.

**Wall**: the obstacle that the player has to destroy to unveil the coin.

**River**: the obstacle that play could not move through it. The river cannot be destroyed by bomb. Bomb can pass through it.

**Castle**: the obstacle that play could not move through it. The castle cannot be destroyed by bomb. Bomb cannot pass through it.

2.2 Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Title** | **Description** | **Priority** |
| FR- 001 | Play New Game | User starts the new game by clicking on the Play button | High |
| FR- 002 | Movement | Tiger player- Uses Up, Down, Left, Right keys to move  Panda player - Uses W, S, A, D keys to move | High |
| FR - 003 | Place Bomb | Tiger player - Uses "L" key to place bomb  Panda player - Uses "F" key to place bomb | High |
| FR - 004 | Map | Wall- Destroyable by bomb; contains hidden treasure  Water - Cannot be destroyed; no player movement possible through it; bomb can pass through it  Castle- Cannot be destroyed; no player movement possible through it; bomb cannot pass through it | High |
| FR - 005 | Bomb | Power 1 - Destroys one brick in each vertical and horizontal directions (up, down, left, right)  Power 2 - Destroys two bricks in each vertical and horizontal directions | High |
| FR - 006 | Mine | Destroys two bricks in each vertical and horizontal directions | High |
| FR - 007 | Lollipop | Increases the bomb power to 2 | High |
| FR - 008 | Score Board | Displays the current score of each player | High |
| FR- 009 | Time monitor | Displays the remaining time of the game | High |
| FR- 010 | Sound effect | Provides sound/music | Medium |
| FR- 011 | Animation effect | Displays animation after explosion | Medium |
| FR - 012 | Game Over | Option to End or Restart the game | High |

Table 1. Functional Requirement

* 1. Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Title** | **Description** | **Priority** |
| NR-001 | Processing time | No single feature should take more than 3 seconds to process any user requests | High |
| NR-002 | Intuitive user-interface | The user interface should be easy to get familiar with and not contain excessive amount of UI elements in single page. The interface should provide the navigation buttons so the user is never stuck on a particular page. | Medium |
| NR-003 | Supportability | The program should be compatible with all desktop operating systems with Java installed | High |
| NR-004 | Legal | BOMB FIGHTER has no age restriction and does not assume any responsibility for issues that may be caused by following its gameplay recommendations. | High |

Table 2. Non-functional Requirements

1. **SYSTEM MODELS**
   1. Use Case Diagram



Figure 3. Use Case Diagram

* 1. Sequence Diagram

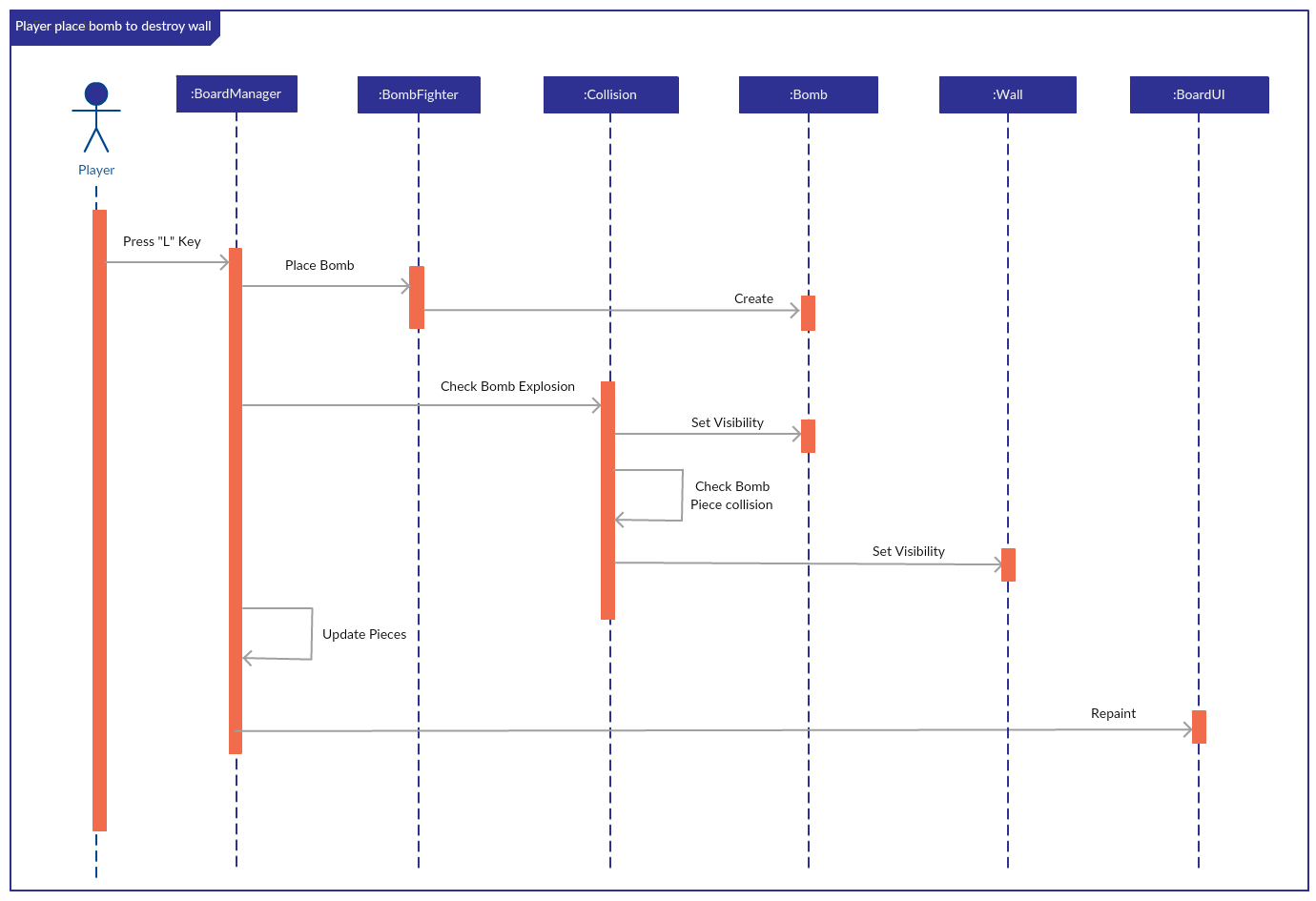


Figure 4. Sequence Diagram that player place bomb to destroy the wall

* 1. Class Diagram

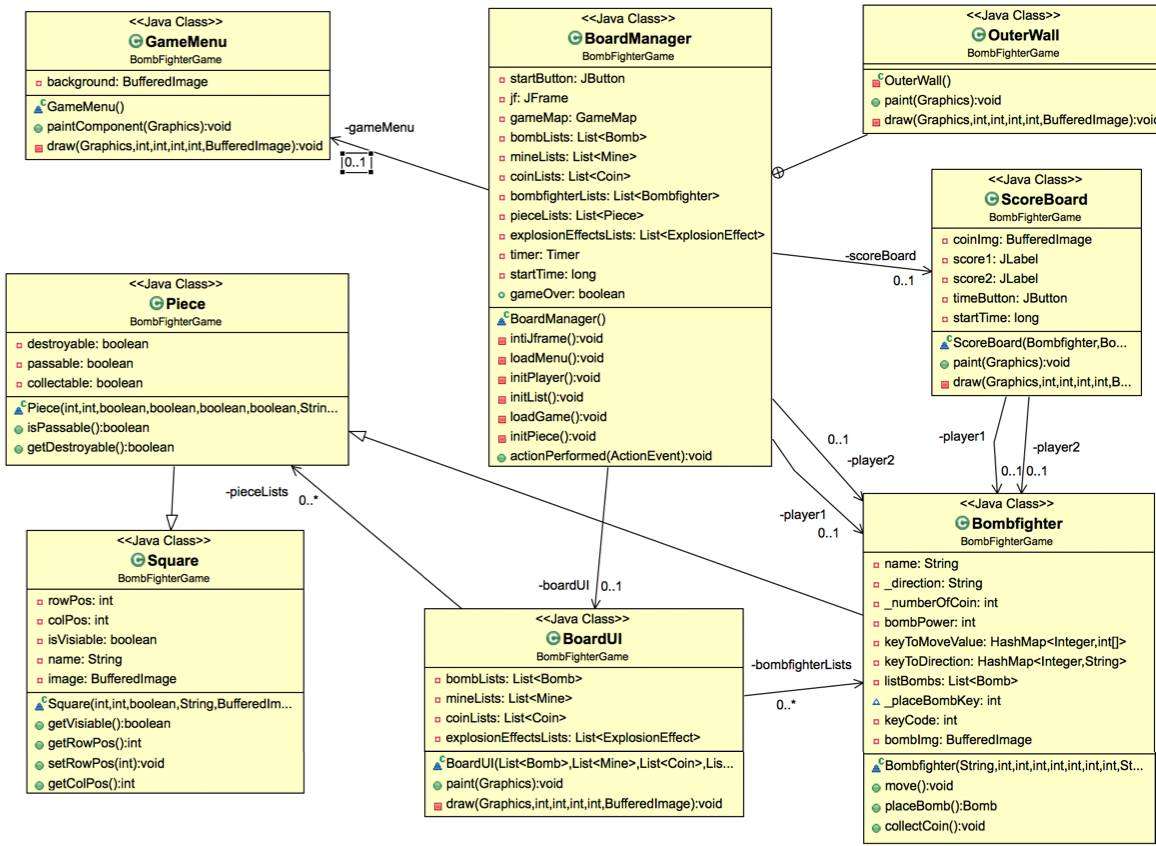


Figure 5. Class Diagram

1. **PROJECT MANAGEMENT**
   1. Traceability

In order to ensure the application is consistent with the requirements that were established before implementation, ID number was assigned to different functional and nonfunctional requirements. The functional and nonfunctional requirements established the key features and inherent qualities of our application. Setting priority levels on each requirement allowed the team to work efficiently so that the final product before demo had a clear focus on core functionalities.In addition, UML diagrams were an integral part of the process as it brought our application to life before we began implementation.After designing the UML diagrams, the requirements were established and understood with far more clarity. The use case diagram was designed to better understand the purpose and functionality. The sequence diagram made it clear which parts of the system played a role in the goal creation workflow. Additionally, using the concept of “separation of concerns”, we were able to design a class diagram that outlined all the separate modules involved in the entire system. Each class was distinct in its purpose, making the codebase extensible and maintainable. Independent modules allowed the system to maximize code reusability because we were able to combine cohesive components into integrated system to leverage their capabilities for recurring usage.

* 1. Testing

The system required implementation of different components that can be written and tested independently before integrating them to the entire system. The 4 components that were able to be tested independently were the Graphic User Interface, the BOMB-FIGHTER movement, the BOMB-FIGHTER coin collection logic, and the BOMB-FIGHTER sound and animation effects. Once the different components of the system passed their independent testing, a collective effort was required to properly integrate them into the system. Regression testing was required to ensure that the system maintained its structural and behavioral integrity after new components were integrated. Once the integration testing and regression testing showed no sign of bugs, the application had then successfully integrated a component, allowing us to move on to integrating the next component and repeating the same process.