SE 3XA3: Module Guide
Return of the Bomberman

Team #18, REM
Miles Jackson jacksa7
Eitan Yehuda yehudae
Ridhwan Chowdhury chowdr11

March 19, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

1 Introduction

Return of the Bomberman is a game designed off of the original Bomber game for the SNES. This project took a simplistic and minimized version of the game and are modifying it to become a fun an interactive game that can be played by multiple players on a web browser. This document is used to show the relations between the SRS document and the implementation. Inside it will outline how and where the requirements are implemented in the code of the game.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The different number of characters a player can choose from.

AC2: The different type and number of power-ups players can use.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Currently, the input solely comes from the keys on a keyboard. The keys allow the user to move their player and drop bombs. If this source of input were to change, the entire design of the game would need to be changed.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Server

M2: Client

M3: GameLoop

M4: Bomb

M5: Player

M6: Explosion

M7: PowerUp

M8: StartGame

M9: EndGame

M10: Movement

M11: PlaceBomb

M12: Collisions

M13: GenerateLevel

Level 1	**Level 2**
Hardware-Hiding Module	Server Client GameLoop
Behaviour-Hiding Module	Player Bomb Explosion PowerUp
Software Decision Module	StartGame EndGame Movement PlaceBomb Collisions GenerateLevel

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the **SRS**. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by *ParnasEtAl1984*. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules ()

Secrets: The data structure and algorithm used to implement the virtual input and output hardware from the user's keyboard.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: Operating System

5.1.1 Server Module (M1)

Secrets: Contains server port and other information about the server.

Services: Creates server to play on and and echos input signals from one player to all of the players in the game.

Implemented By: Return of the Bomberman

5.1.2 Client Module (M2)

Secrets: Contains responses to signals received from the server.

Services: Sends input signals to the server as well as receiving signals from other players.

Implemented By: Return of the Bomberman

5.1.3 GameLoop Module (M3)

Secrets: Contains details about the current state of the game.

Services: Acts as a controller for the game recognizing events that take place and displays them in the game.

Implemented By: Return of the Bomberman

5.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: Return of the Bomberman

5.2.1 Bomb Module (M4)

Secrets: Contains the position, radius and time for detonation of the bomb.

Services: Can be called to generate bomb object on board and calls explosion.

Implemented By: Return of the Bomberman

5.2.2 Player Module (M5)

Secrets: Contains information on a player including location and ID.

Services: Provides a playable character than can be controlled by a user with the help of other modules.

Implemented By: Return of the Bomberman

5.2.3 Explosion Module (M6)

Secrets: Contains radius, animation and central position of an Explosion.

Services: Provides an explosion which can interact with other objects in the game.

Implemented By: Return of the Bomberman

5.2.4 PowerUp Module (M7)

Secrets: Contains the position and type of power-up.

Services: Provides a power-up that can be spawned when a soft wall is destroyed.

Implemented By: Return of the Bomberman

5.2.5 StartGame Module (M8)

Secrets: The signal received to start the game.

Services: Calls Generate Level and starts the game loop.

Implemented By: Return of the Bomberman

5.2.6 EndGame Module (M9)

Secrets: The conditions required for game to be over.

Services: Determines when game is over and stops game loop.

Implemented By: Return of the Bomberman

5.2.7 PlaceBomb Module (M11)

Secrets: The location and radius of a bomb to be placed.

Services: Provides a function to be called to place a bomb when a place bomb signal is received from the server.

Implemented By: Return of the Bomberman

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: Return of the Bomberman

5.3.1 Movement Module (M10)

Secrets: The current location of the player on the board and the current board state.

Services: Enables player movement. Checks player collisions with walls and ensures player does not move through walls.

Implemented By: Return of the Bomberman

5.3.2 Collisions Module (M12)

Secrets: Contains conditions on which a player has deemed to have collided with a Explosion.

Services: Provides a function to check if a play has collided with a Explosion and kills the player if ther did.

Implemented By: Return of the Bomberman

5.3.3 GenerateLevel Module (M13)

Secrets: Contains the template for the game and means to create a fresh game board.

Services: Randomly generates the level to be played filling it with players, soft/hard walls and power-ups.

Implemented By: Return of the Bomberman

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR1	M1, M2, M3, M10
FR2	M1, M2, M3, M10
FR3	M1, M2, M3, M10
FR4	M1, M2, M3, M10
FR5	M1, M2, M3, M4, M11
FR6	M3, M7, M5, M4
FR7	M3, M4, M6
FR8	M3, M4, M5, M7, M11
FR9	M3, M4, M11
FR10	M3, M4, M6, M12
FR11	M3, M4, M6, M12
FR12	M3, M5, M12
FR13	M3, M5, M4, M6, M12
FR14	M3, M5, M9
FR15	M1, M2, M3, M5, M9
FR16	M1, M2, M3, M5, M9

Table 3: Trace Between Requirements and Modules

AC	Modules
AC??	M5
AC2	M7

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. *Parnas1978* said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

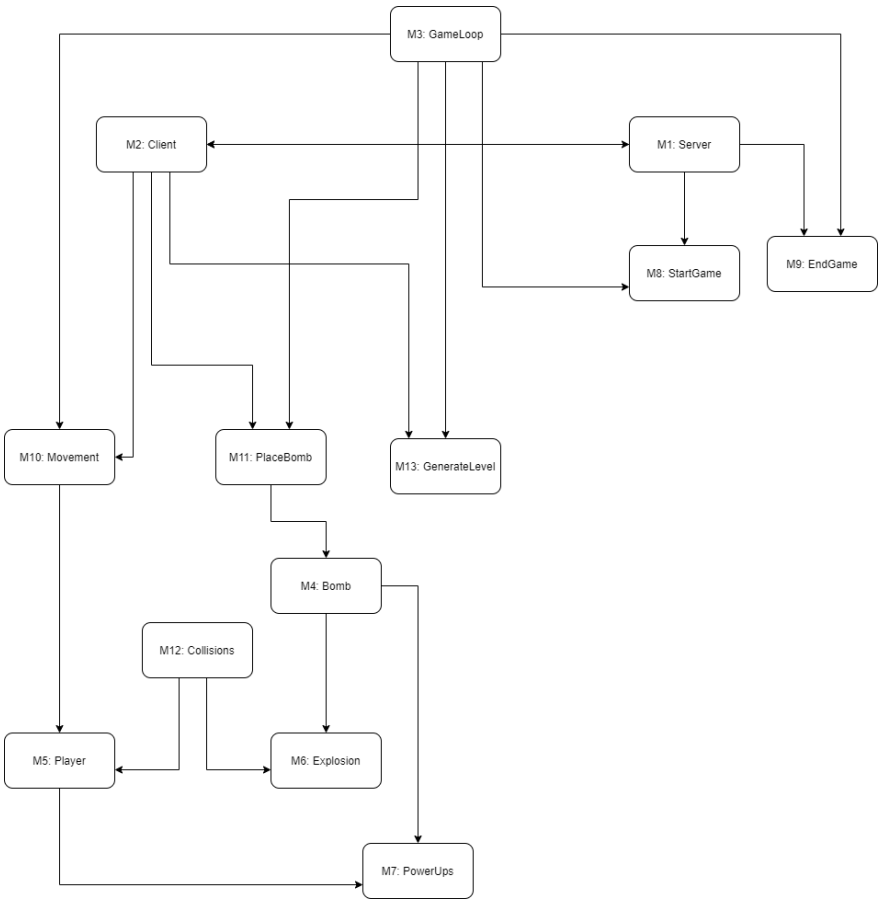

Figure 1: Use hierarchy among modules