SE 3XA3: Module Guide
GardenDefender

Team 29
Ashley Williams, willia18
Declan Mullane, mullanem
Leo Shi, shiy12

November 8, 2018

Contents

List of Tables

List of Figures

1 Introduction

Garden Defender is a re-implementation of a downloadable shooter game to make it accessible via web browser. Through this re-implementation, the project aims to alleviate boredom and provide entertainment on a new platform for a wider range of people.

This module guide document(MG) decomposes s system into modules and ensures they meet the functional and non-functional requirements specified in the software Requirements Specification document(SRS). Meanwhile, a module interface specification document(MIS) is also created to elaborate on the syntax and semantic of modules provided in the MG.

Design principles been used in Garden Defender are information hiding and making the uses relation a hierarchy. Information hiding is applied so that even when the module secret changes, the module design interface does not need to change. These principles also ensure the high cohesion within a module and low coupling between modules.

The document is organized as follows:

- Section 1 gives a brief overview of our project and an introduction to the module design document.
- Section 2 lists anticipated and unlikely changes from Software Requirement Specification document.
- Section 3 summarizes all the modules and gives an overview of the module design.
- Section 4, along with table 3, shows the connection between requirements and design.
- Section 5 provides the module decomposition details including secrets, services and implemented by information for each module.
- Section 6 consists of two traceability matrices. Table 3 shows the connection between modules and requirements. Table 4 connects the anticipated changes to modules.
- Section 7 gives the hierarchy diagram and describes the use relation between modules.

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.

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov. 6th, 2018	1.0	Initial Draft
Nov. 8th, 2018	1.1	Updating All Sections

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 specific hardware on which the software is running.

AC2: The format of the input data for game play may change to accommodate mobile browsers as well as computer browsers.

AC3: The game screen resolution may be adjusted to accommodate both mobile and computer browsers.

AC4: Before final version of game, graphics will likely be modified.

AC5: Level difficulty, length, and features (different enemies and weapons) are likely to be modified.

AC6: The user guide is likely to be updated as levels are updated.

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: Input (game file, keyboard) and Output (game file, screen, speakers) devices.

UC2: There will always be a source of input data external to the software.

UC3: The running environment of the software.

UC4: The algorithm for player's moving and shooting function.

UC5: The winning and losing condition of the game.

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: Hardware-Hiding Module

M2: Input Module

M3: Game Start Module

M4: Player Module

M5: Enemy Module

M6: Projectile Module

M7: Game Status Module

M8: Transition Module

M9: Health Module

M10: Collision Module

M11: Boarder Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module Game Start Module Player Module Enemy Module Projectile Module Game Status Module Transition Module Health Module
Software Decision Module	Collision Module Boarder Module

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 Parnas et al. (1984). 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 (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

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: OS

5.2 Behaviour-Hiding Module

5.2.1 Input Module

Secrets: User input.

Services: Reads user's input from keyboard and mouse.

Implemented By: main.js

5.2.2 Game Start Module

Secrets: Variables, methods, and algorithms used in the game.

Services: Initializes the game, loads all graphics, and displays the game in the web browser using a resolution of 640*480.

Implemented By: index.html, game.js, load.js

5.2.3 Player Module

Secrets: The creation and behavior of the player character.

Services: Creates a player character and controls its movement, aiming, and firing functions.

Implemented By: main.js

5.2.4 Enemy Module

Secrets: The creation and behavior of the enemy characters.

Services: Creates enemies based on current level and controls its movement function.

Implemented By: main.js

5.2.5 Projectile Module

Secrets: The creation and movement of projectile objects.

Services: Creates projectiles, provides them with movement function, and calculates the firing angle using the player character's current targeted direction.

Implemented By: main.js

5.2.6 Game Status Module

Secrets: The algorithms for checking the status of the game.

Services: Tracks player health and position, projectile position, enemy position, and time. Evaluates win/lose conditions.

Implemented By: main.js

5.2.7 Transition Module

Secrets: The transition between states.

Services: Takes the outcome from Game Status Module and display corresponding state image that provide users instruction on what to do next.

Implemented By: main.js, win.js, lose.js, pause.js, transition.js

5.2.8 Health Module

Secrets: Health calculation algorithm.

Services: Calculates and updates health point value; displays it in the health bar region.

Implemented By: main.js

5.3 Software Decision Module

5.3.1 Collision Module

Secrets: The algorithm that determines the behaviour of enemies and projectiles after collision.

Services: Determines the behaviour of enemies and projectiles after collision and gives a signal to related modules including Health Module and Game Status Module.

Implemented By: main.js

5.3.2 Boarder Module

Secrets: The methods for detecting if and controlling the behavior of any object moving out of bounds.

Services: Detects if the player character, enemy character, or any projectile moves out of bounds and provides corresponding handling methods.

Implemented By: main.js

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**
FR-01	M3, M7
FR-02	M3, M7
FR-03	M2,M4
FR-04	M4, M11
FR-05	M4, M6, M7
FR-06	M5,M6
FR-07	M5, M9,M11
FR-08	M5, M6,M10
FR-09	M8,M7
FR-10	M8,M7
FR-11	M8,M7
FR-12	M8
FR-13	M8,M3
NFR-01	M3
NFR-02	M2,M4
NFR-03	-
NFR-04	M1,M3
NFR-05	M1,M3
NFR-06	M1,M3
NFR-07	M1,M3
NFR-08	M1
NFR-09	-
NFR-10	-

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2
AC3	M3
AC4	M5, M6
AC5	M7, M5, M8
AC6	M3

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) 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.

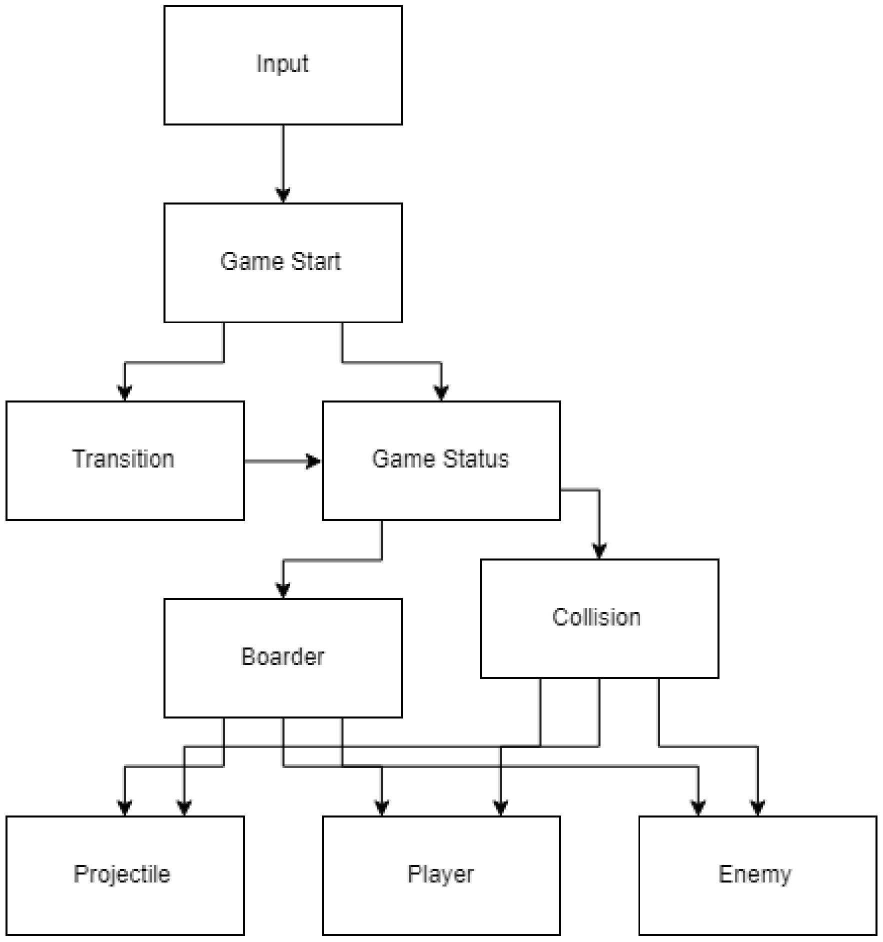

Figure 1: Use hierarchy among modules

References

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.