SE 3XA3: Software Requirements Specification
Asteroids War Game

Team #12, Asteroids War Game
Eric Thai thaie1
Tianzheng Mai mait6
Junhong Chen chenj297
Linqi Jiang jiang121

April 12, 2021

Table 1: **Revision History**

Date	**Version**	**Notes**
March 17, 2021	1.0	Section 1-4
March 19, 2021	2.0	Section 5-7
March 19, 2021	3.0	MIS
April 12, 2021	4.0	Final Revision

Contents

List of Tables

List of Figures

1 Introduction

1.1 Project Overview

Asteroids Multiplayer is based off of the classic arcade game Asteroids with the added functionality of playing with another player. It also refactors the original code to which the user interface is greatly improved to better the game-play experience and user ~~help~~ manual page is also added for better understanding of the game opration rules.

1.2 Document Overview

This document will connect the products functional and non-functional requirements to the technical implementation details including an overview on changes to the system, a module hierarchy, connections between requirements and design, module decomposition, a traceability matrix and a module use hierarchy diagram.

1.3 Document Context

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team. We advocate a decomposition based on the principle of information hiding. This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a

maintainer to update the relevant sections of the document after changes have been made.

- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 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.

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 web browser the software is being run on.

AC2: The hardware the software is being run on.

AC3: The amount of additional players that can play together.

AC4: Additional features such as power-ups or new enemies.

AC5: The different game modes being added.

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 and output devices (standard keyboard and monitor) are unlikely to change.

UC2: The core game purpose (spaceship shooting objects in space while avoiding being hit by other objects/projectiles)

UC3: The method in which 2 players can play the game together (single device)

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: Behaviour-Hiding Module

M3: Software Decision Module

M4: Game Module

M5: Ship Module

M6: Game Canvas Module

M7: Asteroid Module

M8: Menu Module

M9: Alien Module

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.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	M4 M6 M8
Software Decision Module	M5 M7 M9

Table 2: Module Hierarchy

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" . 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 (M2)

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: –

5.2.1 Game Module (M4)

Secrets: ~~How the scores are calculated. How the game is loaded before playing the game. How the game is running when playing the game.~~ The data structure and algorithm used to calculate the user's scores and implement the game.

Services: Handles all the game logic such as algorithm of loading the game, running the game, calculating the scores, etc.

Implemented By: Asteroid project

5.2.2 Game Canvas Module (M6)

Secrets: ~~How the game objects are drawn on the canvas. How the game is displayed on the html web page~~ The data structure and algorithm used to draw the game objects and canvas onto the screen.

Services: Displays the game board and the game objects such as background image, asteroids, spaceship, Aliens, and bullets.

Implemented By: main1.html, main2.html, game1.js, game2.js

5.2.3 Menu Module (M8)

Secrets: ~~How the menu of the game is created. How the menu of the game is formatted and structured. How the buttons link to other web pages.~~ The data structure and algorithm used to create, format and structure the main menu of the game

Services: Displays the menu of the game, formats the text and background image of the game, and provides buttons that allow the user to go to other pages.

Implemented By: index.html

5.3 Software Decision Module (M3)

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: –

5.3.1 Ship Module (M5)

Secrets: ~~How the spaceship interacts with other objects in the game, such as aliens and asteroids. How the spaceship is controlled by the player.~~ The data structure and algorithm used to implement the user's spaceship object in the game, determining how the spaceship interacts with other objects in the game, such as aliens and asteroids and how the spaceship is controlled by the player.

Services: Provides the functionalities of spaceship object.

Implemented By: game1.js. game2.js

5.3.2 Asteroid Module (M7)

Secrets: ~~How the asteroids interact with other objects in the game such as aliens, spaceship and bullets. How the asteroid is broken up after a collision.~~ The data structure and algorithm used to implement the asteroid pbject in the game, determining how the asteroids interact with other objects in the game such as aliens, spaceship and bullets and how the asteroid is broken up after a collision.

Services: Provides the functionalities of asteroid object.

Implemented By: game1.js. game2.js

5.3.3 Alien Module (M9)

Secrets: ~~How the alien interacts with other objects in the game such as asteroids, spaceship and bullets. How the speed and the direction of the alien object are set.~~ The data structure and algorithm used to implement the alien object in the game, determining how the alien interacts with other objects in the game such as asteroids, spaceship as well as bullets and the attributes of the alien object.

Services: Provides the functionalities of alien object.

Implemented By: game1.js. game2.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
R1	M1, M2, M3
R2	M4, M6
R3	M4, M5,M6, M7, M9
R4	M2, M4, M6
R5	M4, M6, M7, M9
R6	M1, M2, M3, M4, M5, M6
R7	M8
R8	M4, M6
R9	M5, M4,M6
R10	M8
R11	M5, M4,M6

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M6
AC2	M1
AC3	M4,M5
AC4	M2,M4, M6,M3
AC5	M4,M7, M9

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. 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.

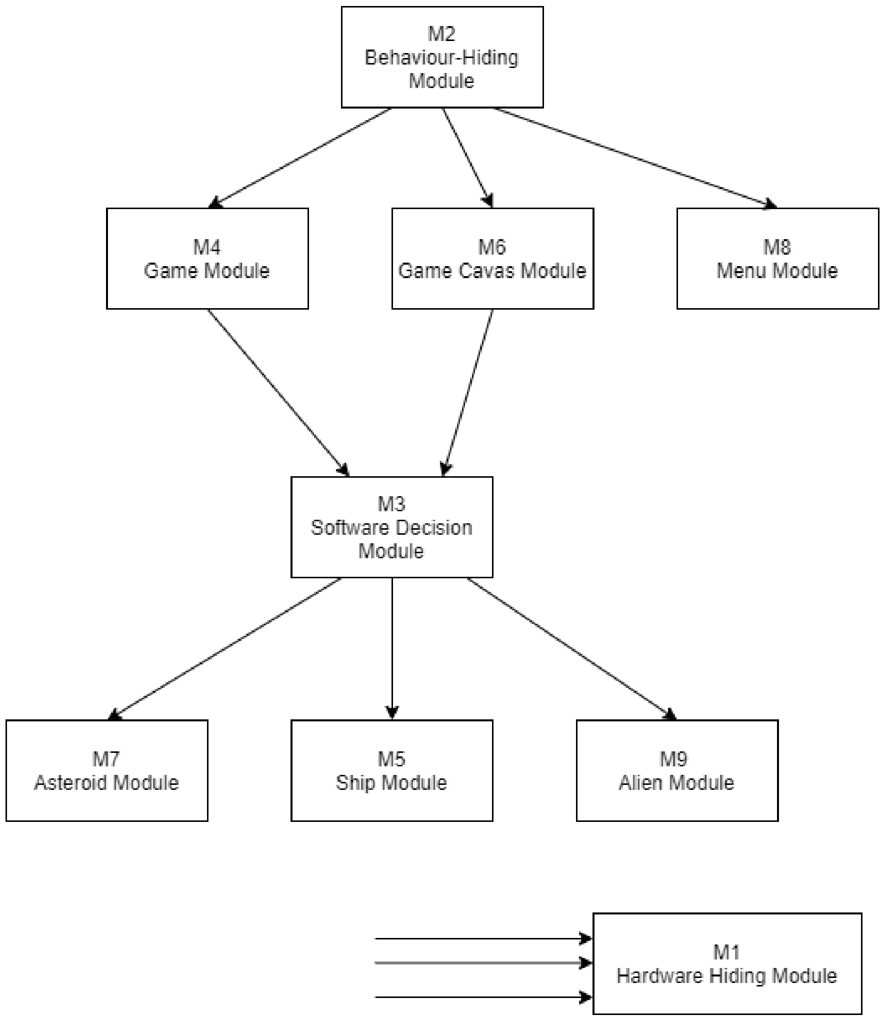

Figure 1: Use hierarchy among modules