SE 3XA3: Module Guide
SpaceShooter

Team #105, SpaceShooter
Dananjay Prabaharan - prabahad
Nishanth Raveendran - raveendn
Hongzhao Tan - tanh10

April 7, 2020

Contents

List of Tables

List of Figures

Date	Version	Notes
March 13, 2020	1.0	Initial Draft
April 6, 2020	2.0	Rev1 Update - Updated overview section and fixed spelling errors within the document.

Table 1: **Revision History**

1 Introduction

1.1 Overview

The SpaceShooter application is a recreation of the 1980s arcade game that enables users to shoot upcoming obstacles (aliens and meteors) with a spaceship. Users could temporarily upgrade their spaceships with power-ups while playing the game. These power-ups include double shooting, shield, missile bullets, and etc. The primary goal of the game is to survive as long as possible.

1.2 Context

The document that follows is the Module Guide of the SpaceShooter application. The purpose of the Module Guide is to demonstrate the deconstruction of the project and how each module serves its respective function/non-functional requirements. The primary intended audience for this document are the new project members, maintainers, and designers. The Module Guide is developed after outlining the requirements on the Software Requirements Specification (SRS). After completing the Module Guide, the next step within the design process is to construct the MIS of the SpaceShooter application.

1.3 Design Principles

Following design principles are incredibly important to any software project to ensure great quality of code and maintainability. When decomposing SpaceShooter into its respective modules, the core design principles that are being followed are (1) encapsulation (2) information hiding and (3) low coupling and high cohesion.

1.4 Document Structure

The remainder of the document is structured as follows:

- Section 2 provides the "Anticipated and Unlikely Changes" of the application.
- Section 3 outlines the module deconstruction (Module Hierarchy) of the application.
- Section 4 outlines the connection between the requirements and modules of the application.
- Section 5 provides the details of each module including its secrets and responsibilities.
- Section 6 provides 2 Traceability Matrices (1) Comparing the design completeness against the application's requirements (2) Connection between modules and anticipated changes.
- Section 7 outlines the module's Uses Hierarchy for the application.

2 Anticipated and Unlikely Changes

This section lists possible changes to the SpaceShooter system. There are anticipated changes which are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

AC1: Program being kept up to date with any new operating systems.

AC2: The user interface and menu operations of the program.

AC3: The player mobility options and controls.

AC4: The addition of new enemy mobs.

AC5: The power up system.

AC6: The addition of new power ups.

2.2 Unlikely Changes

UC1: Input/Output devices (Input: Keyboard, Output: Memory, Screen).

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

UC3: The objective and core game play.

UC4: The score system.

UC5: The health and lives system.

UC6: The art style of all of the game objects.

3 Module Hierarchy

M1: Hardware-Hiding Module

M2: Game State Module

M3: Game Functions Module

M4: Game Objects Module

M5: Constants Module

M6: In-Game Assets Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Game State Module Game Functions Module Game Objects Module
Software Decision Module	Constants Module In-Game Assets Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of SpaceShooter is meant to satisfy the requirements established in the SRS. Each of the requirements are satisfied by a single or several module(s) with related information that are decomposed from the system. The Game State Module uses functionalities from all other modules but not the Hardware-Hiding Module to ensure the system can be correctly executed. The appearance, usability and game-state-related requirements are satisfied by Game State Module, image and audio files that are used in the implementation which provides a UI for users to operate the system and make correct respond on the UI due to requests from the users or state of the game.
The system is written in Python which satisfies the requirement that the system shall be able to run on the mainstream operating systems. Maintainability requirements are satisfied by proper modularization and documentation in the implementation of the system. The connection between requirements and modules is listed in Table 3.

5 Module Decomposition

The following modules are decomposed to David Parnas' principle of information hiding. They are broken down in the following matter, Secret, Service, and Implemented By. Secrets will describe in a single word what it is that the module is hiding. Services will detail what it is the module does, and Implemented By states by what means the module is implemented. However, if the entry is OS, this signifies that the following module is provided by the operating system.

5.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithms that supports the basic functionalities of users' hardware.

Services: Provides an interface between the hardware and SpaceShooter.

Implemented By: OS

5.2 Behaviour-Hiding Module

5.2.1 Game State Module (M2)

Secrets: The internal game data of the player and the game state.

Services: Controls and updates the state of the game. This module will switch screens when the player pauses or dies. During the "Playing Game" state, this module will spawn meteors and aliens, checking if the player touches a meteor or power up, and update the game score, player health, and power up state. The module also invokes the game functions to draw the graphical output.

Implemented By: gameState.py

5.2.2 Game Function Module (M3)

Secrets: The general functions that are used in the main running loop of the game.

Services: Generates main menu, pause menu and death menu when the game is in "Pre-Game State", and "Paused State" respectively; Displays the texts that are supposed to be shown on the UI with specified font and size; Displays the "shield bar" and "lives" which graphically acknowledge the user about the condition of the in-game spaceship when the system is in "Playing Game State"

Implemented By: gameFunctions.py

5.2.3 Game Object Module (M4)

Secrets: The respective properties and methods of each critical object in the game.

Services: Defines the properties of each interactive object within the game (Missile, Bullet, Explosion, Player, etc); Establishes various object methods against different scenarios in the game.

Implemented By: gameObjects.py

5.3 Software Decision Module

5.3.1 Constants Module (M5)

Secrets: The values of all constants that are used in the implementation

Services: Records the values of the width and height of the general UI screen, frame rate of the game, length and height of the UI for in-game spaceship's "shield bar", and the value of tuples which define colors of the general UI.

Implemented By: constant.py

5.3.2 In-Game Assets Module (M6)

Secrets: Data of each asset utilized in various stages of the game.

Services: Denotes the path to the assets being used in the game; Assigns and establishes which asset file to use for each object in the game.

Implemented By: InGameAssets.py

6 Traceability Matrix

Req.	**Modules**
R1	M1
R2	M3
R3	M3
R4	M2, M3
R5	M2, M3
R6	M2, M3
R7	M2, M3
R8	M3, M4
R9	M2
R10	M4
R11	M2
R12	M2
R13	M4
R14	M4
R15	M2, M4
R16	M2, M4
R17	M2, M4
R18	M2, M4
R19	M2, M3, M4

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M3
AC3	M4
AC4	M4
AC5	M2
AC6	M4

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. For two modules A and B, A uses B means that it might be necessary for A to have access to a correct implementation of B, so that A can provide only desired functionalities to its clients. Figure 2 illustrates the use relation between the modules..

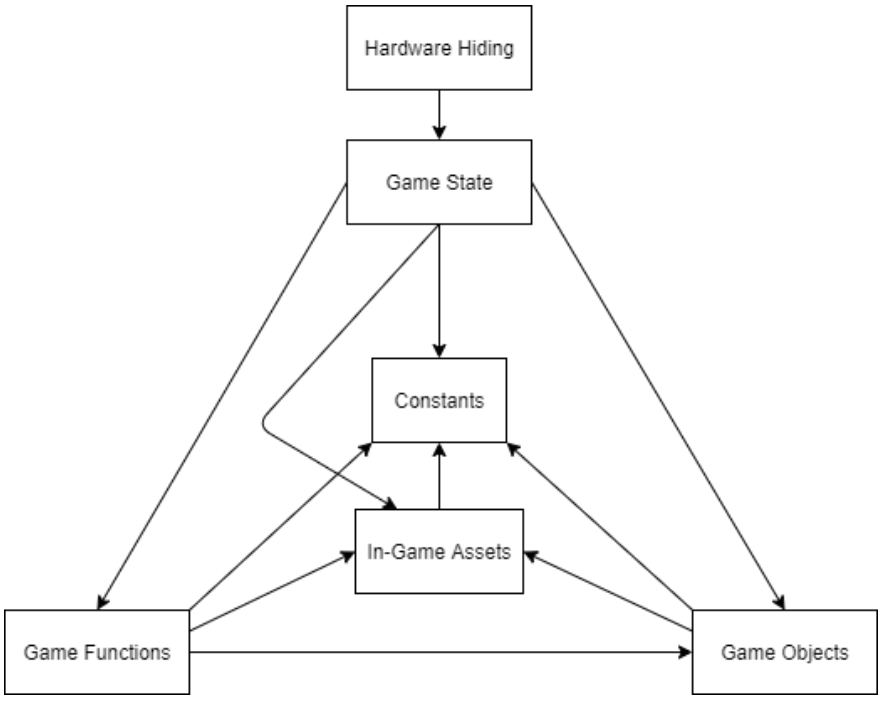

Figure 1: Use hierarchy among modules

8 Gantt Chart

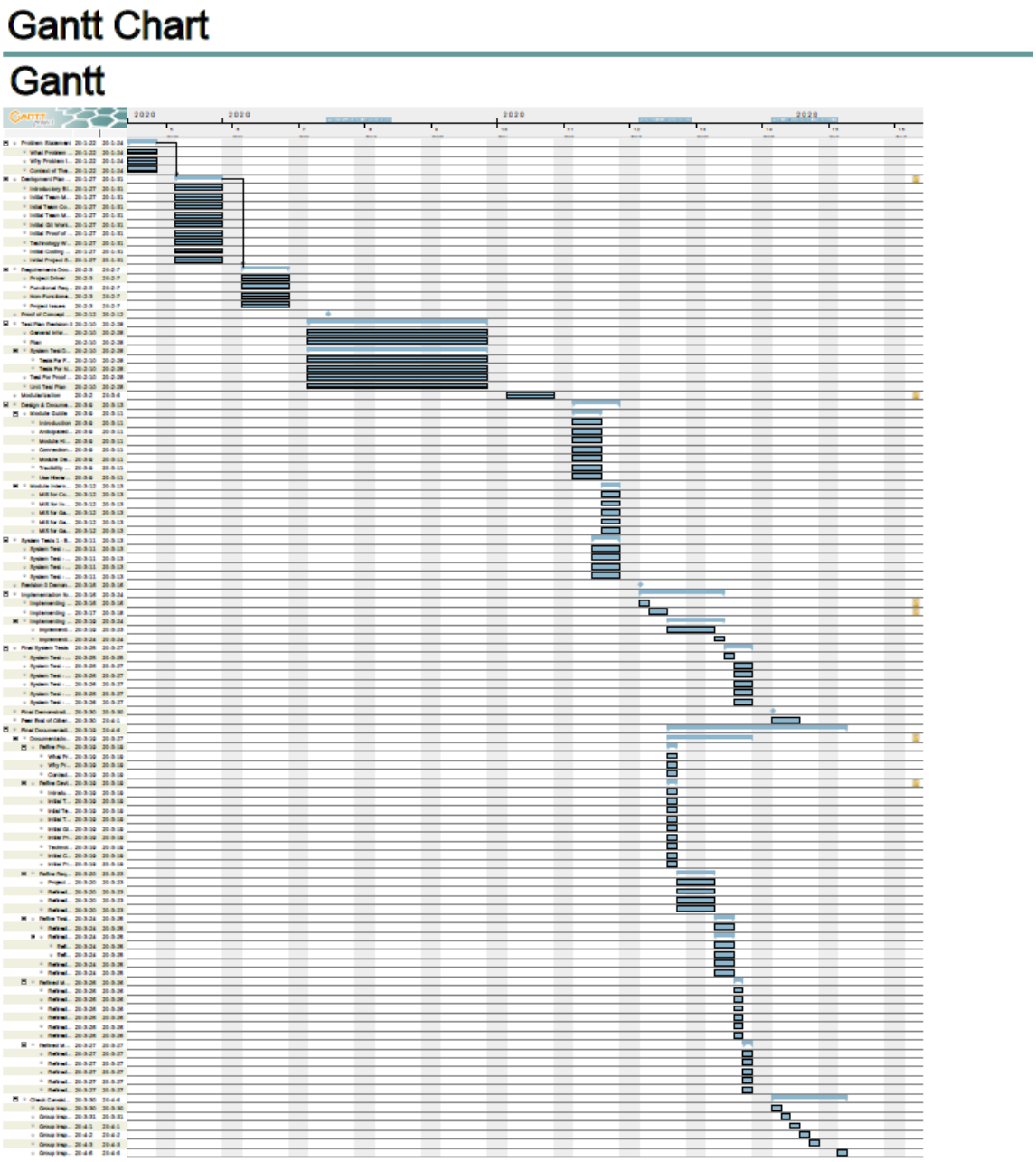

Figure 2: Gantt Chart