SE 3XA3: MG
Space Invaders

March 18, 2022

Team Information:

Team Number	Name	MACID
L03 G07	Qianlin Chen	chenq84
	Jiacheng Wu	wuj187
	Tingyu Shi	shit19

Table 1: Revision History

Date	**Developer(s)**	**Change**
January 26, 2022	All team members	Initial Document
March 18, 2022	Qianlin Chen	Section3, section5
March 10, 2022	Jiacheng Wu	Section1, Section2, Section5, Section6
March 10, 2022	Tingyu Shi	Section4, Section7

Contents

List of Tables

List of Figures

1 Introduction

Space Invader is a video game that player will control an aircraft to eliminate all the monsters. The goal of the project is to upgrade the graphics of the game. Additionally, we will use software engineering principle to redevelop it.

1.1 Purpose

The purpose of this document is to demonstrate the modular representation of the system in order to separate the concerns and hide the information. The document also provide high cohesion and low coupling to reduce the complexity of the system.

1.2 Scope

The Module guide will provides all the modules that based on the requirements in the SRS document. Also, the external behaviours of the modules will go into the MIS documents.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software requirement specification
PC	personal computer
FPS	frams per second
GUI	graphic user interface
HP	health point of the aircraft

Table 3: **Table of Definitions**

Term	**Definition**
Space invader	The name of a video game
Graphic User Interface	a graphic representation for users to interact
Score	The number that represents the achivement of users
Players	the users of the game that follow certain rules to succeed
Aircraft	the users will controll the aircraft to play the game
Monster	the goal of the game, the target the aircraft to attack.
items	the element in game that can boost users' ammo

1.4 Overview of Document

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 data structure to represent the monster.

AC2: The algorithm to determine the collision between bullets, monsters, obstacles and the aircraft.

AC3: The visual effect when collisions happen.

AC4: How the scores are calculated.

AC5: Soundtrack of the game.

AC6: The algorithm to represent shooting bullets.

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/Output devices: The game only focuses on keyboard as input and screen as output.

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

UC3: The software architecture of the game since the PAC is the best option for this design.

UC4: The running environment of the game.(WIN,MACOS,LINUX)

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 4. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: SingleController

M2: DoubleController

M3: TotalController

M4: MonsterDisplay

M5: SpaceShipDisplay

M6: MonsterMatrixDisplay

M7: BulletDisplay

M8: ScoreDisplay

M9: ObstaclesDisplay

M10: AmmoDisplay

M11: HeartDisplay

M12: BombDisplay

M13: SetUpDisplay

M14: Monster

M15: SpaceShip

M16: MonsterMatrix

M17: Bullet

M18: Score

M19: Obstacle

M20: BulletState

M21: MonsterColor

M22: Ammo

M23: Heart

M24: Bomb

M25: CollisionDetection

M26: Driver

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	SingleController DoubleController TotalController MonsterDisplay SpaceShipDisplay MonsterMatrixDisplay BulletDisplay ScoreDisplay ObstaclesDisplay AmmoDisplay HeartDisplay BombDisplay SetUpDisplay Driver
Software Decision Module	Monster SpaceShip MonsterMatrix Bullet Score Obstacle BulletState MonsterColor Ammo Heart Bomb CollisionDetection

Table 4: 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 5.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas 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

N/A

5.2 Behaviour-Hiding Module

5.2.1 Driver(M26)

Secrets: Actions to do to inilialize the total controller.

Service: Initialize the total controller.

Implemented By: Space Invader

5.2.2 SingleController(M1)

Secrets: Actions to be done after the user select single mode.

Service: Once single mode is selected, the user will play with one aircraft in the game.

Implemented By: Space Invader

5.2.3 DoubleController(M2)

Secrets: Actions to be done after the user select double mode.

Services: Once double mode is selected, the users will play with two aircraft in the game.

Implemented By: Space Invader

5.2.4 TotalController(M3)

Secrets: Actions the system do to provide single mode and double mode for users.

Services: Allow the users to select between single mode and double mode.

Implemented By: Space Invader

5.2.5 MonsterDisplay(M4)

Secrets: The method to visualize a single monster.

Services: Display a monster on the screen and update the visualization as its model changes

Implemented By: Space Invader

5.2.6 SpaceShipDisplay(M5)

Secrets: The method to visualize the aircraft.

Services: Display the aircraft on the screen and allow users to controll and display the change after user giving inputs.

Implemented By: Space Invader

5.2.7 MonsterMatrixDisplay(M6)

Secrets: The method to visualize over one monsters.

Services: Display a number of monsters on the screen and update their visualization as their model change.

Implemented By: Space Invader

5.2.8 BulletDisplay(M7)

Secrets: The method to visualize bullets.

Services: Display bullets on the screen and update their visualization as their model change.

Implemented By: Space Invader

5.2.9 ScoreDisplay(M8)

Secrets: The method to visualize scores.

Services: Display scores on the screen and update their visualization as their model change.

Implemented By: Space Invader

5.2.10 ObstaclesDisplay(M9)

Secrets: The method to visualize Obstacles.

Services: Display Obstacles on the screen and update their visualization as its model change.

Implemented By: Space Invader

5.2.11 AmmoDisplay(M10)

Secrets: The method to visualize ammo items.

Services: Display ammo items on the screen and update their visualization as its model change.

Implemented By: Space Invader

5.2.12 HeartDisplay(M11)

Secrets: The method to visualize Heart items.

Services: Display Heart items on the screen and update their visualization as its model change.

Implemented By: Space Invader

5.2.13 BombDisplay(M12)

Secrets: The method to visualize Bomb items.

Services: Display Bomb items on the screen and update their visualization as its model change.

Implemented By: Space Invader

5.2.14 SetUpDisplay(M13)

Secrets: The method to display welcoming message, game mode selection message, game instructions and game addiction prevention message.

Services: Display welcoming message, game mode selection message, game instructions and game addiction prevention message.

Implemented By: Space Invader

5.3 Software Decision Module

5.3.1 Monster(M14)

Secrets: Data structures used to represent a monster and algorithms to change its states.

Services: Provide methods to represent a monster.

Implemented By: Space Invader

5.3.2 SpaceShip(M15)

Secrets: Data structures used to represent the aircraft and algorithms to change its states.

Services: Provide methods to represent the aircraft.

Implemented By: Space Invader

5.3.3 MonsterMatrix(M16)

Secrets: Data structures used to represent a number of monsters.

Services: Provide methods to represent over one monsters.

Implemented By: Space Invader

5.3.4 Bullet(M17)

Secrets: Data structures used to represent bullets and algorithms to change their states.

Services: Provide methods to represent bullets.

Implemented By: Space Invader

5.3.5 Score(M18)

Secrets: Algorithm to calculate the scores.

Services: Provide methods to determine the calculation of scores.

Implemented By: Space Invader

5.3.6 Obstacle(M19)

Secrets: Data structures used to represent Obstacles and algorithms to change their states.

Services: Provide methods to represent Obstacles.

Implemented By: Space Invader

5.3.7 BulletState(M20)

Secrets: Data used to represent bullet state.

Services: Provide different states for bullets.

Implemented By: Space Invader

5.3.8 MonsterColor(M21)

Secrets: Data used to represent monster color.

Services: Provide different colors for monsters.

Implemented By: Space Invader

5.3.9 Ammo(M22)

Secrets: Data structure used to represent ammo item and algorithms to represent its usage.

Services: Provide methods to represent ammo items and actions to change the ammo of the aircraft.

Implemented By: Space Invader

5.3.10 Heart(M23)

Secrets: Data structure used to represent heart item and algorithms to represent its usage.

Services: Provide methods to represent heart items and actions to change hp of the aircraft.

Implemented By: Space Invader

5.3.11 Bomb(M24)

Secrets: Data structure used to represent bomb item and algorithms to represent its usage.

Services: Provide methods to represent bomb items and actions to change states of monsters/obstacles.

Implemented By: Space Invader

5.3.12 CollisionDetection(M25)

Secrets: Algorithms to dectect if two objects in game overlap

Services: provide methods to determine if two objects collide

Implemented By: Space Invader

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-FR3	M13
FR4	M8, M18, M13
FR5	M15, M13
FR6	M13
FR7	M6, M13
FR8	M10, M11, M12, M13
FR9	M5, M13
FR10	M19, M13
FR11	M18, M13
FR12-FR14	M13
FR15	M8
FR16	M23, M15
FR17	M3
FR18-FR22	M16
FR23-FR27	M14
FR28	M4
FR29	M19
FR30-FR31	M22, M23, M24
FR32	M16, M14, M24
FR33	M22, M5, M7
FR34	M23, M5
FR35	M22, M23, M24,, M16
FR36-FR39	M3
FR 40	M15

Table 5: Trace Between Requirements and Modules

AC	**Modules**
AC1	M14
AC2	M26
AC3	M4, M6
AC4	M18
AC5	M3
AC6	M17

Table 6: 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.

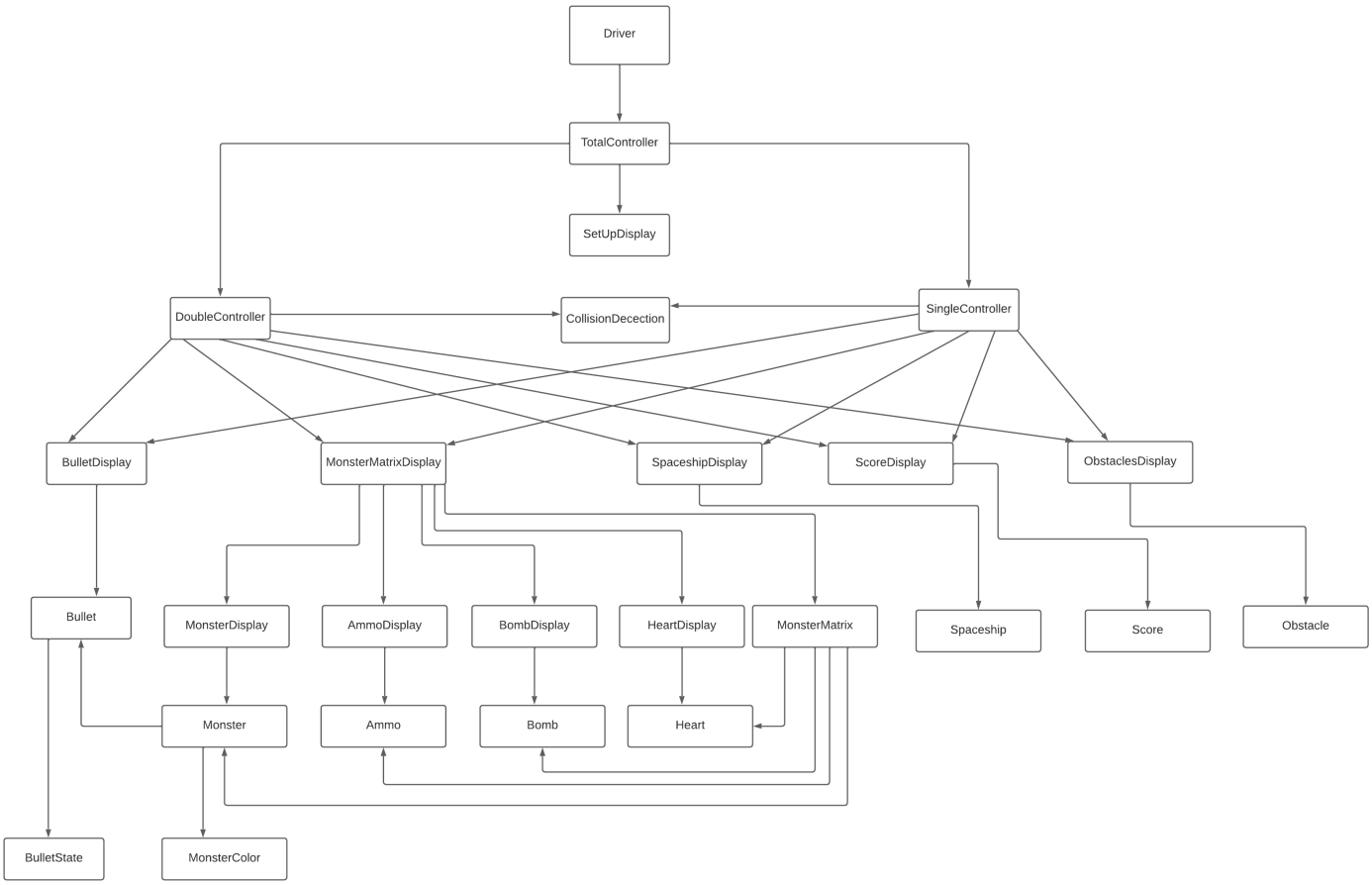

Figure 1: Use hierarchy among modules

8 Gantt Chart

For the gantt chart of testing plan, please access it by the following links:
.gan file
.pdf file

Reference

David L. Parnas. On the criteria to be used in decomposing systems into modules. Coom.ACM, 15(2):1053-1058, December 1972.
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.