SE 3XA3: Module Guide
Spaceshooter Remix

Team #4, IRS Development
Ibrahim Malik maliki2
Ryan Schnekenburger schneker
Saad Khan khans126

December 5, 2018

Contents

List of Tables

List of Figures

Table 1: Revision History

Date	**Version**	**Notes**
5 November 2018	Rev 0	Authored by Ibrahim, Saad, Ryan
December 5, 2018	Rev1	Updated document and added the changes recommended by TA.

1 Introduction

1.1 Overview

The project is re-implementation of the classic space shooter game that was popular in the 1980s and 1990s. There are several versions of this game online, but we attempt to create it for a computer using Python 3. This game will require a keyboard and will hopefully provide a retro feel to the user which will create a nostalgic effect. The objective of the game is to survive for as long as possible by destroying or avoiding the asteroids.

1.2 Context

There are 2 specific documents in this design and document specification. This document is the Module Guide (MG) and there is also Module Interface Specification (MIS) document.

The Module Guide was based on the original Software Requirements Specification (SRS) which outlined exactly the purpose and the main objectives of this project. There were several functional and non-functional requirements that were outlined in that document that detailed the expectations for the game. This document will provide a modular decomposition of the implementation of the system. It will show how this game can be executed to meet both functional and nonfunctional requirements. It will display the architectural design of this system describing the modules, the rationale for decomposition, the relationship between the modules and any constraints that will need to be respected.

The MIS document is based off the Module Guide. It will describe the semantics of each module and include assumptions, state variables, environment variables, uses, access routines, and exceptions. Every function or method will be described in this manner showing the inputs and outputs and provide a basis of how each module is executed. It is an example of a detailed design.

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 (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). 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 laid out by Parnas et al. (1984), 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.

This document will be helpful for the following people:

- 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, the feasibility of the decomposition, and flexibility of the design.

1.3 Design Principles

1.4 Document Structure

The document is organized as below:

- 2 will list the unlikely and anticipated changes of the software requirements.
- Section 3 will summarize the module decomposition and present a hierarchy separating different decision modules
- 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 demonstrates how the modules are connected to each other.

2 Anticipated and Unlikely Changes

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

2.1 Anticipated Changes

Anticipated changes refer to the source of information that is within the modules and how changing that will affect the system. Changing one of the anticipated changes will require only one module being changed that will hide the associated decision. This allows for versatility for change without majorly impacting the rest of the project and is called design for change.

AC1: The hardware or computer on which the software is running.

AC2: Algorithms for controlling the speed, power or the movement of the sprites

AC3: The images of the game objects can be changed. This includes game objects like characters and asteroids. There will likely be an increase in the number of objects.

AC4: There will be a change in the gameplay to allow for increased difficulty of the gameplay as game time progresses

AC5: Additional menus providing instructions or credits is likely

2.2 Unlikely Changes

The module design should be as general as possible. Nevertheless, a general system can be more complex. To avoid risking modification of several modules at once, these changes are unlikely.

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

UC2: Input data will always come externally from the software

UC3: The python 3 implementation of the game

UC4: The purpose of the game and the objectives of destroying or avoiding asteroids

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: ~~Player Module~~

M2: Constants module

M3: Spawn module

M4: Asteroids module

M5: Animations Module

M6: Destroy module

M7: IRS_Space_Shooter Module

M8: Player_Control

M9: Player_Hide

M10: Player_Move

M11: Player_Shoot

M12: Overheat_Control

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Player_Hide constants module Spawn module Asteroids module Player_Move
Software Decision Module	Animations Module Destroy module IRS_Space_Shooter Module Overheat_Control Player_Control Player_Shoot

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The system is designed to satisfy the requirements that were outlined in the SRS. The system is decomposed to modules and the connections between requirements and modules are listed in Table 3. The appearance and usability requirements will be satisfied with the input module which creates the initial environment to allow the user to interact within. That will be the game window. Then the requirements of the actual gameplay involving the movement of the players and the asteroids will be created by the Animations and IRS_Space_Shooter module. This module will also affect the precision, reliability, capacity and environmental

requirements. The player, spawn and asteroid module will contribute to the appearance requirement as well. The game will be easy for the user to install because instructions will be given clearly and the user will just have to execute the game using a simple python command. Most computers have python compilers and this will satisfy the cross-platform ability of the game. All the commands and language used in the input and output modules will be clear and avoid any political or cultural violations. There is no data stored in any of the modules and hence no risk of private user information being a security risk.

5 Module Decomposition

5.1 Hardware Hiding Modules

5.2 Behaviour-Hiding Module

Secrets: Updating the sprite array, and calculating asteroid damage

Services: Constantly updates the sprite array when anything dies, and also deals damage to spaceship according to asteroid size

Implemented By: M7

Secrets: Visibility of the Player Sprite

Services: Hides the player sprite upon death, and has it reappear once the game starts again

Implemented By: M9

Secrets: Spawning sprites for missle and power ups

Services: Adds sprites to groups that are drawn, when missles are shot or powerups are dropped

Implemented By: M3

Secrets: Decides movement speed of player

Services: Adds values to the player movement parameters, allowing the player to move side to side

Implemented By: M10

5.2.1 Input Format Module

Secrets: Taking in sounds and images

Services: Takes in images and sounds files and transforms them into useable forms

Implemented By: M7

5.3 Software Decision Module

Secrets: Displays all non-gameplay sprites

Services: Draws all sprites such as lives, health, and menu

Implemented By: M5

Secrets: Deals with death animations

Services: Once a sprite dies, this module changes the image to an explosion and removes the sprite from the group

Implemented By: M6

Secrets: Runs the Game

Services: Runs the main game loop, and calls for updates for all sprites

Implemented By: M7

Secrets: Controls the overheat mechanic

Services: Keeps track of how far along the player is on the overheat bar, and if the player overheats, doesnt allow the player to shoot

Implemented By: M12

Secrets: Player values

Services: Keeps track of all player values, and send commands to other modules for updates.

Implemented By: M8

Secrets: Controls shooting

Services: Controls shooting for the player, and also sends updates to the overheat module to keep track of the overheat bar

Implemented By: M12

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	M6
R2	M6, M5
R3	M5
R4	M5
R5	M7, M3
R6	M7
R7	M6
R8	M7
R9	M1
R10	M7
R11	M1
R12	M1
R13	M7, M5
R14	M1

Table 3: Trace Between Requirements and Modules

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

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section we have a uses hierarchy wherein A will need B to execute but, B can execute without the use of A. The uses relation can been translated right from python by observing the import statement and where there exists an import statement that is a uses relation between that module and the import statement. Of course since we are creating a game each of our modules use pygame however I did not include that since it is a pre-created library. This is of the form of a directed acyclic graph (DAG) where you may start from any

of the nodes and take a path but you will not be able to return to the node that you began on from each node.

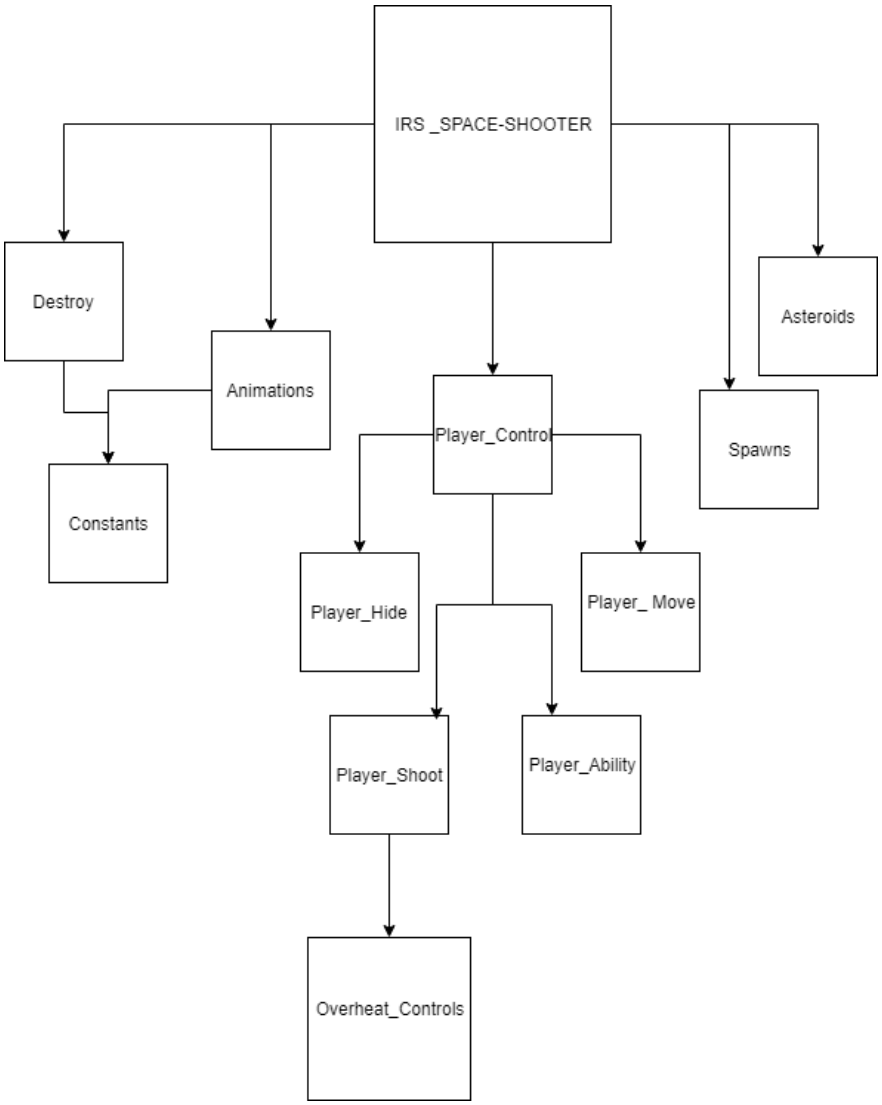

Figure 1: Use hierarchy among modules

8 Functional Requirements Reference

R1: When the game is executed in python a new window shall be opened
Fit Criterion or Test Case: Is a new window opened after the program is executed?

R2: The game shall open a main menu once executed in python
Fit Criterion or Test Case: check to see if there is a main menu when you first execute the game

R3: A loading screen will be displayed once 'enter' is pressed when the main menu is displayed
Fit Criterion or Test Case: press enter when on the main menu and see if a loading screen is displayed

R4: Score shall increase when an asteroid comes in contact with a bullet
Fit Criterion or Test Case: check to see if the score increases when a bullet comes in contact with an asteroid on the game screen

R5: Asteroids shall be removed from the game display once shot
Fit Criterion or Test Case: When in game and not in the main menu or the loading screen, check to see if when a bullet comes in contact with an asteroid that asteroid is no longer on the game screen

R6: The health bar shall decrease if the ship comes in contact with an asteroid
Fit Criterion or Test Case: check to see if the health bar decreases once when hit by an asteroid on the game screen

R7: The ship shall move to the left when the left arrow key is hit
Fit Criterion or Test Case: press the left arrow key and check if the ship moves left on the game screen

R8: When the 'r' key is hit, all scores will be cleared
Fit Criterion or Test Case: Check that the score is 0 once the 'r' key is hit while on the game screen

R9: When the spacebar is tapped one bullet is fired
Fit Criterion or Test Case: press the spacebar and check to see if a bullet is fired when on the game screen

R10: When the right arrow key is pressed the ship shall move to the right
Fit Criterion or Test Case: press the right arrow key and check if the ship moves to the right when on the game screen

R11: When the 'q' button is pressed when the main menu is displayed the window shall close
Fit Criterion or Test Case: When the main menu is displayed press the q key and check to see if the window displaying the main menu is still open

R12: When the health bar is empty a life is removed from the game screen
Fit Criterion or Test Case: Check to see if the amount of lives is one less than it was when the health bar was not empty on the game screen

R13: When the player has no lives and their health bar is empty the main menu should be displayed
Fit Criterion or Test Case: Check to see if the main menu is displayed once the health bar is empty and there are no lives remaining on the game screen

R14: When the health bar is zero the ship shall be temporarily removed from the screen
Fit Criterion or Test Case: Check to see if the ship is still displayed on the game screen when the health bar is empty

References

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

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.