SE 3XA3: Software Requirements Specification
Staroids

Team 20, Staroids
Moziah San Vicente, 400091284, sanvicem
Eoin Lynagh, 400067675, lynaghe
Jason Nagy, 400055130, nagyj2

November 7, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Oct 29/18	0.1	Added basic information
Nov 7/18	0.15	Added anticipated changes
Nov 7/18	0.2	Added some parts of module decomposition
Nov 7/18	0.25	Added unlikely changes and module breakdown
Nov 7/18	0.3	Added tracability matrix
Nov 7/18	0.35	Completed tracability matricies
Nov 7/18	0.4	Removed unnessesary sections
Nov 7/18	0.41	Instantiation of text and formatting

1 Introduction

Staroids is decomposed into many modules, as is the 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. Staroids is developed with the knowledge that modifications are frequent and is developed to best support these modifications in future.

Our design follows the rules layed out by **?**, 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 Staroids, the Software Requirements Specification (SRS), the Module Guide (MG) were 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 **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** 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 **??** describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes Staroids. There are two categories for changes based on the likeliness of the change. Anticipated changes are listed in Section **??**, and unlikely changes are listed in Section **??**. Anticipated changes are planned or probable in the forseeable future of Staroids and unlikely changes are changes that are not planned or forseen for the life of Staroids.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules of Staroids. Each change will only require alteration of one module that contains the relevant hidden information. The approach adapted here is called design for change.

AC1: Staroids must be kept up to date with any new operating systems or updates to the supported internet browsers are released.

AC2: Enlarging of the playing screen and the scale of all on screen objects.

AC3: Player firing and destruction sounds.

AC4: Collision detection between all on screen sprites.

AC5: Relative speed of all projectiles (both player and alien shot)

AC6: Speed, locomotion, size and spawning of the alien

AC7: The shape and size of asteroids

AC8: Amount of lives given to the player when the game starts.

2.2 Unlikely Changes

Certain elements of Staroids are designed in a way that avoids unessesary complexity and as a result, would be less likely to require alteration. Other elements of Staroids are complex and are standard throughout the source code, so alteration of that element would require the editing of many module. Hence, it is not intended that these decisions will be changed.

UC1: Controls for the player and menu operations.

UC2: The sound module's method of playing sounds and controls over existing sounds.

UC3: The update structure of all in game objects and how they operate.

UC4: The destructive property of the large and medium asteroids into 3 asteroids of one size smaller.

UC5: The state structure of the core game.

UC6: How the states transfer into one another.

UC7: Method of text dislaying to the screen.

3 Module Hierarchy

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

M1: Hardware-Hiding Module

M2: Head Module

M3: Utilities Module

M4: Sound Module

M5: GameObject Module

M6: GameState Module

Level 1	**Level 2**
Hardware-Hiding Module	M1
Behaviour-Hiding Module	M2 M5 M6
Software Decision Module	M3 M4

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of Staroids is intended to satisfy all requirements established in the SRS. This lead to Staroids being separated into seperate modules that contain contain related information. The connection between requirements and modules is listed in Table **??**.

5 Module Decomposition

Modules of Staroids are decomposed according to the principle of "information hiding" proposed by **?**. 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 JavaScript libraries.

5.1 Hardware Hiding Modules (M??)

Secrets: How the game handles user keyboard input, how the canvas is outlined.

Services: Loads scripts onto canvas, therefore giving the user a visual reference for the game to then decide how they would like to interact with it.

Implemented By: index.html

5.2 Behaviour-Hiding Module

5.2.1 Sound Module (M??)

Secrets: How sounds are played, how audio files are accessed, and the states of each audio object.

Services: Controls a plethora of options for each sound that the game might need including: muting, unmuting, pausing and unpausing, stoping, and checking if the sound is paused.

Implemented By: sound.js

5.2.2 Utilities Module (M??)

Secrets: The values of all constants for the game, and variables to handle user inputs of pressed keys.

Services: Funtions to test for if a key is pressed. The game object handles score, lives, asteroids, as well as getters and setters for all game object attributes.

Implemented By: utilities.js

5.2.3 Head Module (M??)

Secrets: All data shown on the screen will be pulled from this file.

Services: Combines all scripts together so they are able to access each others methods.

Implemented By: head.js

5.3 Software Decision Module

5.3.1 GameObject Module (M??)

Secrets: Contains base game object, along with all other secondary objects which inheret from it. This includes all the attributes specific to each object.

Services: Getters and setters for all game objects, draw functions to put them to the canvas, and interaction functions between objects such as collisions.

Implemented By: gameobject.js

5.3.2 GameState Module (M??)

Secrets: Containts state machine for the game as well as the main running loop of game.

Services: Sets game objects to desired values based on states, as well as updates the game though the main loop as time elapses and user inputs are sent in through the event listeners defined in this module. This module however does not process the inputs, only listens for them and then alerts the utilites module to process them into variables which can then be recognized.

Implemented By: gamestate.js

6 Traceability Matrix

Below shows the traceability matrices for Staroids. These track which requirement is fulfilled by which module and which modules would need to be changed for each anticipated change.

Req.	Modules
R1	M??, M??
R2	M??
R3	M??
R4	M??
R5	M??
R6	M??
R7	M??, M??, M??
R8	M??, M??
R9	M??
R10	M??
R11	M??, M??
R12	M??
R13	M??
R14	M??
R15	M??
R16	M??
R17	M??
R18	M??
R19	M??, M??, M??
R20	M??, M??, M??
R21	M??, M??, M??
R22	M??, M??, M??
R23	M??, M??

Table 3: Trace Between Requirements and Modules

AC	Modules
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between all Staroids modules is provided. **?** 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 **??** illustrates the use relation between the modules.

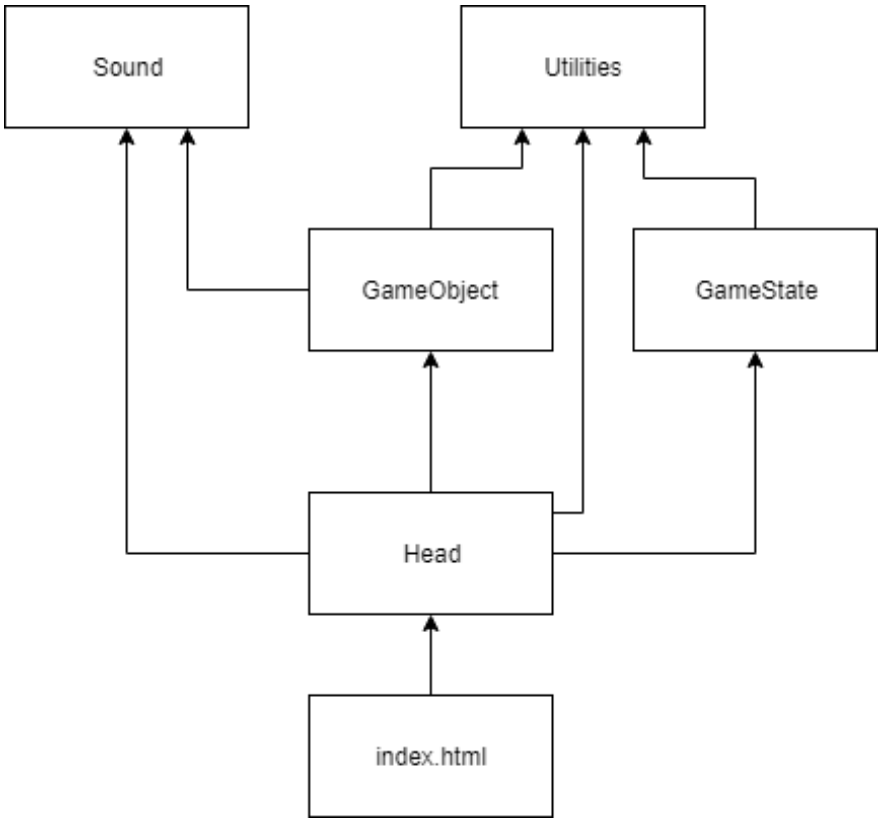

Figure 1: Use hierarchy among modules