SE 3XA3: Software Requirements Specification
Staroids

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

November 9, 2018

Contents

List of Tables

List of Figures

1 Introduction

The Staroids Project is a re-development of the classic arcade game Asteroids, based on an open source implementation found on github. It is meant to be run on all major browsers hence why it was chosen to be written in JavaScript. Throughout the re-development the main principles that have been kept in mind are modularity, and information hiding. The modularization of Staroids will be explained further thorughout this document, and the principle of information hiding is touched on more in the MIS document, and is shown through the use of getters and setter to access state variables of objects, instead of directly being able to change them. The Module Guide is designed to specify the modular structure of the staroids project and as well relate them to goals set out in the SRS document. It should allow both future designers and maintainers to easily identify the different aspects of the software. The document is organized starting with the 2 Changes section that lists both anticipated and unlikely changes of the software requirements. The 3 Module Hierarchy summarizes the module decomposition as constructed by the Staroids team, keeping in mind any anticipated changes that may come in the future. The 4 Connection section specifies the connections between the software requirements of the project as layed out in the SRS documents, and the modules as listed below. The 5 Module Decomposition gives a detailed description of each module, for further specification please refer to the MIS. The 6 section includes the two tracability matricies, one to check the completeness of the design and modules against the the requirements in the SRS, and the other to show the relations between anticipated changes and the corresponding modules that they would affect. The final section 7 describes the uses relations between the modules of the project.

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 2.1, and unlikely changes are listed in Section 2.2. Anticipated changes are planned or probable in the foreseeable

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

future of Staroids and unlikely changes are changes that are not planned 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 unnescessary 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 displaying 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 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: 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	M3 M4 M5 M6
Software Decision Module	M2

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 leads to Staroids being separated into separate modules that contain related information. The connection between requirements and modules is listed in Table 3.

5 Module Decomposition

Modules of Staroids are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). 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 (M1)

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 (M4)

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, stopping, and checking if the sound is paused.

Implemented By: sound.js

5.2.2 Utilities Module (M3)

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

Services: Functions 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 GameObject Module (M5)

Secrets: Contains base game object, along with all other secondary objects which inherit 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

5.2.4 GameState Module (M6)

Secrets: Contains 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 utilities module to process them into variables which can then be recognized.

Implemented By: gamestate.js

5.3 Software Decision Module

5.3.1 Head Module (M3)

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

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	M1, M6
R2	M6
R3	M6
R4	M6
R5	M6
R6	M6
R7	M6, M5, M3
R8	M5, M3
R9	M5
R10	M5
R11	M5, M6
R12	M5
R13	M5
R14	M5
R15	M5
R16	M6
R17	M5
R18	M5
R19	M5, M5, M4
R20	M5, M5, M4
R21	M5, M5, M4
R22	M5, M5, M4
R23	M5, M4

Table 3: Trace Between Requirements and Modules

AC	Modules
AC2	M3
AC2	M3
AC3	M4
AC7	M5
AC7	M5
AC7	M5
AC7	M5
AC7	M5
AC8	M6

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. 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.

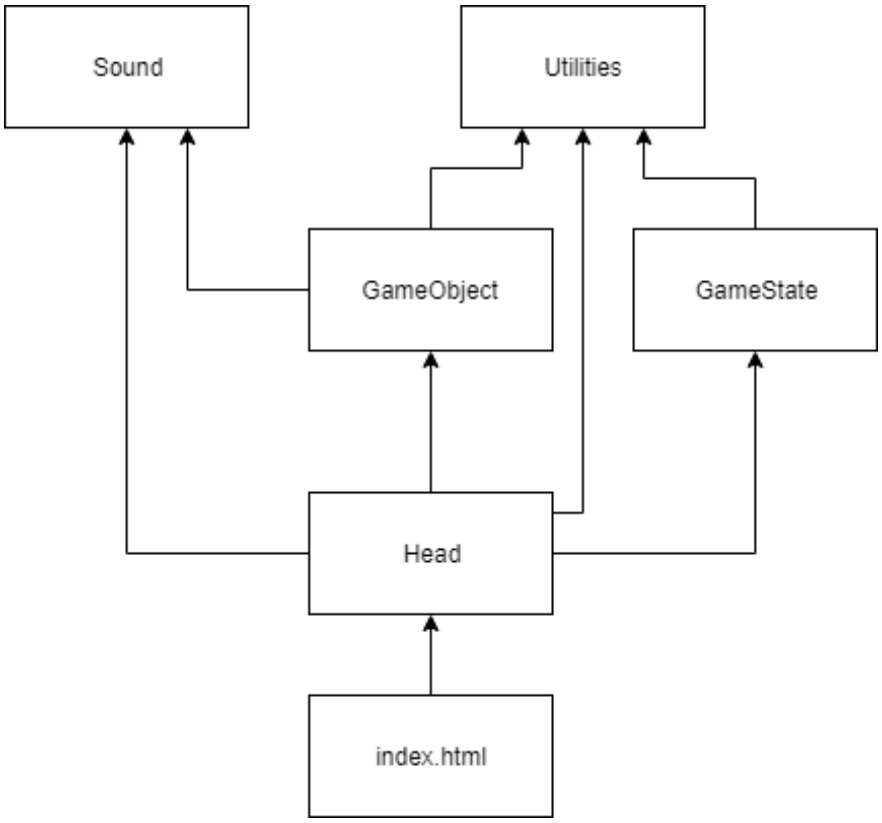

Figure 1: Use hierarchy among modules

References

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.