SAPIENZA
UNIVERSITÀ DI ROMA

Master of Science in Engineering in Computer Science

Interactive Graphics

Project
Interstellar Horizon

Riccardo Vecchi

Student ID: 1467420

vecchi.1467420@studenti.uniroma1.it

28/06/2018

Contents

1 Project specification

1.1 Practical work

Create an **interactive and graphics application**.

The theme can be chosen, but it needs the professor's approval.

It can be done in **groups of 1 to 3 persons** (exceptionally 4).

You can use «basic» **WebGL** or advanced libraries, such as **ThreeJS** (`http://threejs.org/`) or **Babylon** (`http://babylonjs.com/`) or others.

You can use models created with a modeler or found on-line. You cannot import animations.

The project MUST include:

- **Hierarchical models**
  - At least one and more complex of the model used in homework2;
- **Lights and Textures**
  - At least one light, textures of different kinds (color, normal, specular, ...);
- **User interaction**
  - Depends on your theme, as an example: turn on/off lights, change viewpoint, configure colors, change difficulty, ...;
- **Animations**
  - Most objects should be animated, in particular the hierarchical models should perform animations that exploit their structure. Animations cannot be imported, should be implemented by you in javascript (WebGL, ThreeJS or other approved library).

The project must be uploaded on the **GitHub Classroom repository** (`https://classroom.github.com/g/zK7SwNrf`) and should be runnable by activating a GitHub Page.

1.2 Report

The accompanying document for the presentation of the project, **should be both a technical presentation and a user manual** and contain:

- Description of the environment used (basic WebGL or other);
- List of all the libraries, tools and models used in the project but not developed by the team;
- Description of all the technical aspects of the project;
- Description of the implemented interactions;
- The length is up to you, at least 5-10 pages.

2 Project implementation

2.1 Introduction

Interstellar Horizon is a **3D browser game**, in which the user **can drive a spacecraft in an attempt to survive and gain points**. The user can encounter several type of threats like asteroids, black holes and so on. The project is substantially a **WebGL program** developed with **all standard technologies** like HTML5 and JavaScript.

2.2 Requirements

For running the game, **there are only 3 requirements**:

1. a modern web browser that supports WebGL and standard technologies (namely Google Chrome, Mozilla Firefox and so on);
2. an internet connection for downloading the site's files and the dependencies (Three.js and stats.js);
3. a GPU enough powerful (not too old) to running smoothly the game.

Considered the application's target, I optimized as much as I could.
I tested the performance with a very low-end Intel integrated GPU (4^{th} generation) and **I managed to mantain 60 FPS for the most part of the time at resolution FullHD**. With a 2K monitor (and 2K resolution), my game is almost always over 30 FPS.

About the compatibility, I noted that in Google Chrome the game is much smoother in respect, for instance, Microsoft Edge or Mozilla Firefox. In the case of Firefox and Edge, I think there is a problem in the WebGL engine of the browser.

Unfortunately, for external causes (independent from me), I have to say that the user is strongly adviced to use Google Chrome.

Beyond pure performance, the game has the same behaviour among all browsers tested. Indeed, although the Javascript behaviour is (in theory) standardized, sometimes can arise differences that I have removed.
New file formats, like WebP, although more efficient in terms of space, are not supported by all browsers, hence **I opted for the max compatibility**.
For the personalized cursor icon, I have used 2 images formats, .cur (for compatibility with Internet Explorer and Edge) and .png (for all browsers).

2.3 Structure and code

The **structure of the project** is the following:

game/
- game.js
- game.min.js
- interstellar_horizon.html
- lensflare.js
- server.cmd
- **images/**
  - (list of images in different formats)
- **models/**
  - (list of JSON models)
- **sounds/**
  - (list of flac files)

The file **interstellar_horizon.html** is the web page of the game and prepare all what is needed if the user clicks the button "Start game". **It contains also CSS rules (internal style)**, to describe graphically the homepage itself and some animations inside the game.
Instead, **game.js** and **game.min.js** are the effective implementation of the game, and the latter **is the obscured and minified version**.
The script **server.cmd** is used for running a local Python HTTP server, avoiding several problems that can arise when there is a loading of a model or a texture from a file system (due to browsers security policies). If the project is hosted in a real web site, this file can be removed.

Images and other media in general have been downloaded from the web, paying attention in those without copyright or with a

too restrictive license.

An interesting resource for free sounds is `https://freesound.org/`.

For the models, I found several web sites, including `https://clara.io` that, after signing up, enables to download many 3D models in several formats as the **JSON format** supported by Three.js.

I found useful also the **official editor of Three.js**, reachable at https://threejs.org/editor/. It allowed me to modify 3D models in an intuitive and graphical way, without the (unthinkable) manual revision of the JSON files. After the changes, I exported the files in the same format as the original without problem.

As **external libraries** I used:

- **Three.js**, a cross-browser JavaScript library and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser. It is based on WebGL and handles all the low-level aspect of the standard graphics library.
- **stats.js**, which is a useful library that provides statistics like FPS (Frames per Second), allocated memory (in megabyte) and so on. This library was essential in the debug phase, but I choose to mantain it also in the final release, because it still deliver useful information.

Both **external libraries are not in the local folder of the project, but they will dinamically downloaded on the Internet by CDN (Content Delivery Network) links**.
During the development of the game, several versions of Three.js have been released, hence I cyclically updated to the latest version, taking care that the game continued to work as want. Currently, **Three.js is at Revision 93,** and is the version used in the game.

Initially, I used also the **jQuery** library for simplify repetitive commands and have an higher-level of control on the JavaScript code. Afterwards, I preferred to delete jQuery dependencies, because I noted that this would have unnecessarily burdened the project.

For the same reason, considered the nature of the game, **for collision detection and the detection of the laser hit, I preferred to use the raycaster technique**, provided by the APIs of Three.js instead of importing and using a physics library like **Physijs**.

In the writing of the code, I followed two main paradigm: **procedural** and **objects**. Some features, as the generation of new obstacles (objects) is better implemented in a procedural way, while the spaceship and its characteristics are better described by an object approach.

I found the model of the spaceship on the site `https://clara.io` and, with the appropriate API, I imported the model in the application. The spaceship's shape is inspired from the famous Star Wars saga and is constructed as a **hierarchical object** (it can be viewed easily thanks to the Three.js editor). I modify the model with the editor and I did other optimizations that will be presented later in the Optimization section.

I complicated the 3D Model of the spaceship adding **two children object**: the shield (it is visible only during collision) and the propulsor of the ship.

The shield is basically a sphere drawed with the **wireframe technique**.

The propulsor instead, is a **lensflare object**. I used a library found in the examples of Three.js that I slightly modified and, instead of dinamically download it, I put the lensflare.js file in the local folder.

The 3D model of the spaceship is put inside a more general Javascript object "spaceship" that contains all the parameters and behaviour of the object, for instance the maximum life, the speed of the vessel, the damage of the laser and so on.

The other most important Javascript object is **gameStatus**, that, in addition to the options of the game (like the maximum number of asteroids) will retain also useful information for the gamer, like the current points.

Collisions are calculated by raycaster objects. There can be **two types of collision**:

1. the spaceship's laser hit an asteroid;
2. the spaceship itself intersects with an object.

In the first case, the function **generateLaser()** (that is fired with a click), will create a raycaster object, paying attention of **translating correcly the screen coordinates to the 3D world coordinates**.
The second case is simpler, because we don't need to translate numbers between different system coordinates. The main difference between the two types of collision is that, in the latter, we will have a different behaviour depending on the type object hit.

Basically, the game has **three types of objects**:

- **asteroids** → they are sphere geometries in which I apply a randomly appropriate texture. If the laser hit one of them, they lose life, and if the life reaches 0, the asteroid will be destroyed and the user will gain points. Instead, if the first kind of collision occurs, will be the spaceship to lose life.
- **helpers** → they are cubes with different colors that if caught up helps the user, giving life, munitions or fuel for warp speed;
- **black holes** → a sphere geometry that will attract every object to its gravitational center. If the spaceship will collide with a black hole, the match is immediately lost. Black hole are immune to laser firing (the collision will not be calculated) There can be only a black hole at time.

Animations that is worth mentioning are the following:

- **The movement of the spaceship**. I created a complex system and a queue events array to enable the spaceship to move diagonally and for making its movements as fluid as possible.
- When the laser hits an asteroid, I simulated an **explosion** with a particle system. The class that permits this kind of object/animations is **THREE.Points**, where in previous version of the library, this class was named **THREE.ParticleSystem**. I mantain max only two particle systems at a time for performance reasons.
- **The "fire" of the spaceship's propulsor**. It will oscillate between different light intensity to look more real. In the "warp speed" mode it will bright more than double respect to the original value.
- The **attraction of the black hole**.
- **Almost all of the events in the game will fire an appropriate sound**, and I implemented several functions to manage them easily. The user, indeed, can choice to mute all the sounds or return to the initial condition.

During the development of the game, I noted that the function **animate()** that updates and moves all the game world was not very suited for the goal, indeed, the dynamic of the world was FPS-dependent. Namely, on a platform where the game runs at 20 FPS instead of 60, the objects will move three times slower. Hence, I created a function called **tick()** that, independently of the FPS will be called at interval of **30 milliseconds** and updates the world. **Thanks to this decoupling, the dynamic of the game is much more independent from the platform used**. Indeed the game will evolve always in the same way, also if the platform experiments a slowdown on the frame per second produced.

Obviously, **the game handles the resizing of the window**, recalculating the projection matrix and setting the renderer correctly. Moreover, the spaceship is free to go off-screen, but there is a routine that will "attract" slowly the object to the screen for avoiding strange behaviours.

Initially, all images of the interface like the life bar or the mute symbol was implemented with **THREE.Sprite** objects that enable to stick an image "before the camera", but have some limits and are less versatile respect to a combination HTML and CSS. Moreover, thanks to this technique, it is possible to offload and let the THREE.js engine to focus on the real core of the game.

As soon as the project was growing in size, it became essential to implement a **load manager** that handle all the loader object and loading procedures, **before** the script's game ran. I it weren't so, giving the **asynchronous nature of the loading procedure**, the game could crashes at the start.

After a game over, will appear an opportune "splash screen" in which the gamer can choice between quit the game and restart a new match. **The procedure for restarting the game is very complex**, and includes the cleaning of the scene (avoiding insidious memory leaks), and the cautious reinitialization of objects in the right order (I tried also of avoiding to deallocate and reinitialize object that could be maintained).

Finally, for avoiding the same pattern inside a match or among different matches, most parts of the game, like the generation of obstacles or the type of helper that will spawn are random (pseudo-random).

2.4 Optimization

Given that the project is suitable to become a real online game, I optimized it in several ways.
First of all, **I compressed every type of media** (images, sounds and so on) in lossless or lossy way (depending on the format and the possible space gain) with programs like FileOptimizer and, for bigger files also in a lossy way (although still invisible to human's eye).
Moreover, I noted that the JSON model was too numerical precise for my goals. Hence, **I reduced it's precision to 2 decimal digits**, through the following Notepad++ regular expression:

```
(?<=\d\.\d{2})\d+
```

that takes all numbers after the second decimal digit and I substituted them with the empty string. This gave me another 2MB saved.

Moreover, I used some tools online for minimizing the javascript code. Obviously in this work the minified version is not directly presented for evaluation purpose.

Instead, for the external library Three.js, after the debugging phase, I switched to the minified version.

All this optimization have shrunk my project of about 50-60% in size.

Obviously, the most important thing is the algorithm optimizations. I tried to remove all bottlenecks and make the game as efficient as possible.

For instance, after the adding of the propulsor effect, I lost about 10 FPS. To fill the performance gap, I tried to remove another bottleneck of my code: whenever the user destroyed an object or the object went out of the screen, the object were deallocated and a new object were created. With the developer tool of the browser, I noted that the garbage collector took a significant portion of the time. Hence, I followed the **object pool pattern**, namely I created "pools array" that will be used as a sources for "new objects" and as sinks for objects to delete. Thanks to this strategy, I only switch elements between an array with objects used currently and an array with objects ready for enters the game, but most important, the engine avoid many allocation and deallocation and the garbage collector took much less time.

Wherever possible, I replaced the geometries of the mesh objects with **buffer geometries** that is an optimized version of the geometry, although harder to manage. **Internally Three.js uses only buffer geometries**, hence, if the geometry is already of the right type, there will be no expensive conversions.

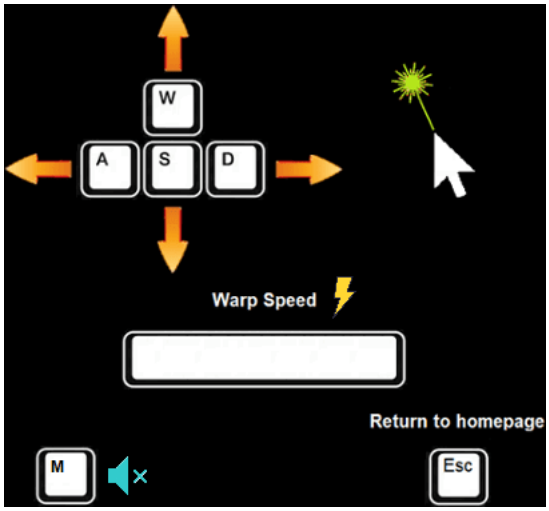
3 User manual

For selecting the "right commands" and interactions, **I used my past experiences** in videogames and **I listened some people for advices**. Fortunately, I noted that the **learning curve for understanding the interactions is very low**.

The game is meant to be played with laptop or desktop computers and **not** by mobile devices like smartphones or tablets.

The homepage of the game (interstellar_horizon.html) presents two buttons, namely "Start Game" and "Instructions".
By clicking Instructions will appear an image that intuitively explains which type of interactions the user can perform and with which commands.

The image that appears in the homepage is the following:



Briefly, **the user must interact with mouse and keyboard**.

The mouse will be used as a pointer for the laser cannon and by clicking one of the mouse buttons or the wheel, if there is enough laser charge, the spaceship will fire a laser ray that will travel toward the coordinates (x,y) pointed by the mouse. The z axis depends on the distance of the first object hit by the ray.

The keyboard commands available and relative functions are the following:

- keys **W, A, S, D** → this classical buttons can be used to move the spaceship up, left, down and right respectively;
- key **Space** → while is mantained pressed, if there is enough warp-speed charge, then the spaceship will travel at light speed;
- key **M** → enable or disable music and sounds;
- key **Esc** → exit the game and return to homepage;
- key **R** → if in game over, this button restarts a new game (it is also described in the game over splash screen).

The game continues until the life of the spaceship will be greater than 0.

Ther user will face **three types** of object:

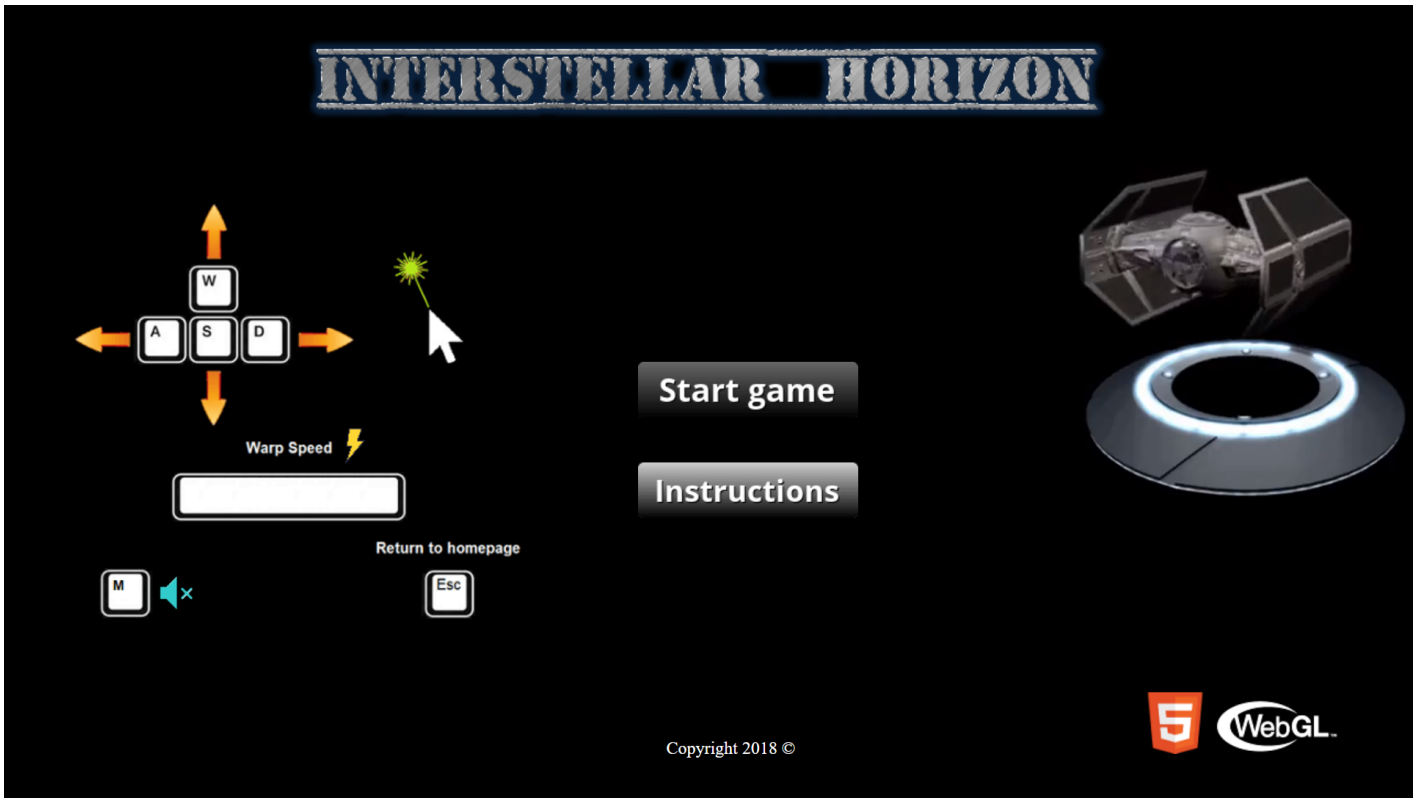
- **asteroids** → if an asteroid is destroyed the user will receive 1000 points, but if an asteroid collides with the spaceship, then the latter will be damaged;
- **helpers** → cubes with different colors that if caught up helps the user, giving life, munitions or fuel for warp speed;
- **black holes** → one of the worst enemy in the universe that when appears, will attract every object to its gravitational center. If the spaceship will collide with a black hole, the match is immediately lost.

Be parsimonious with the use of the warp speed! Sometimes is the unique weapon to escapes from a giant black hole!

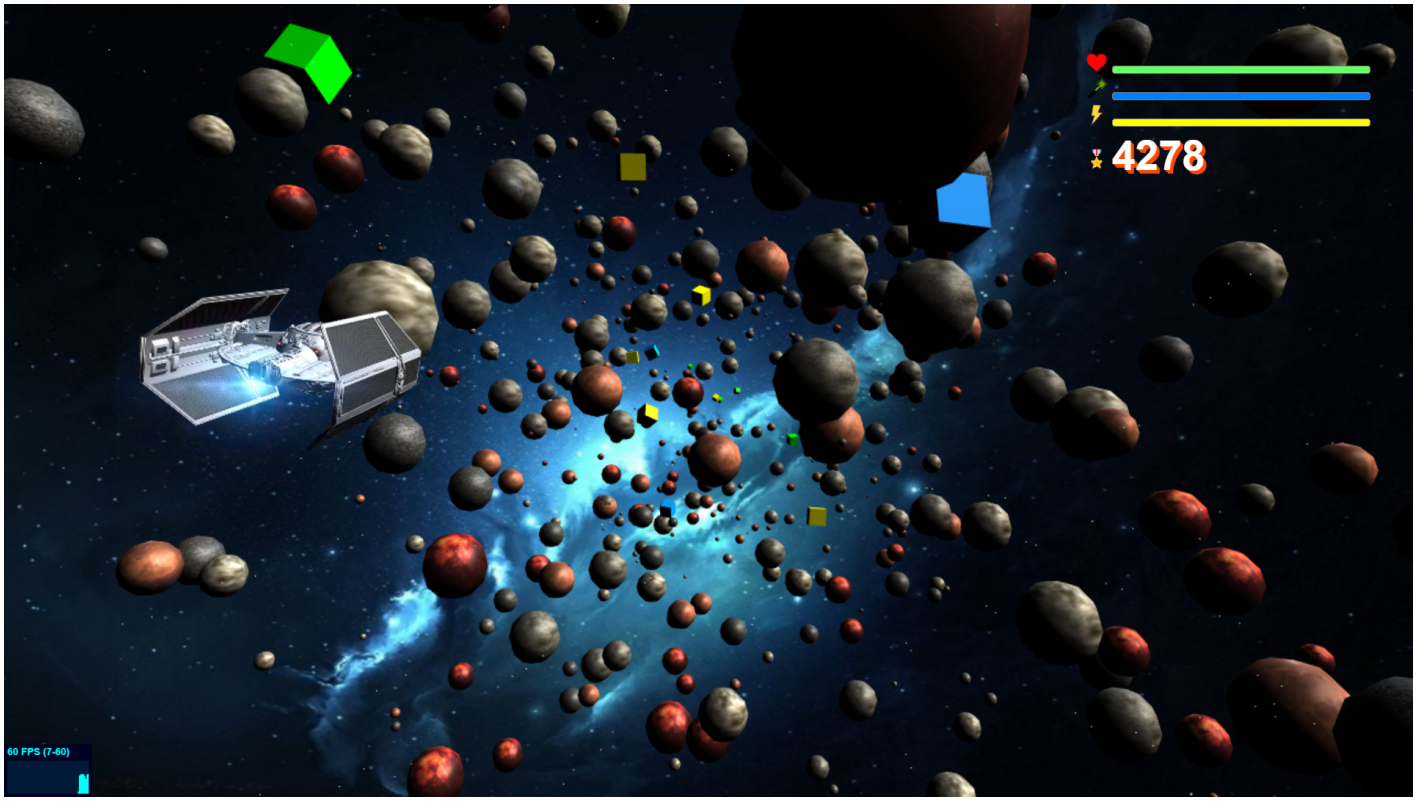
Try to survive!

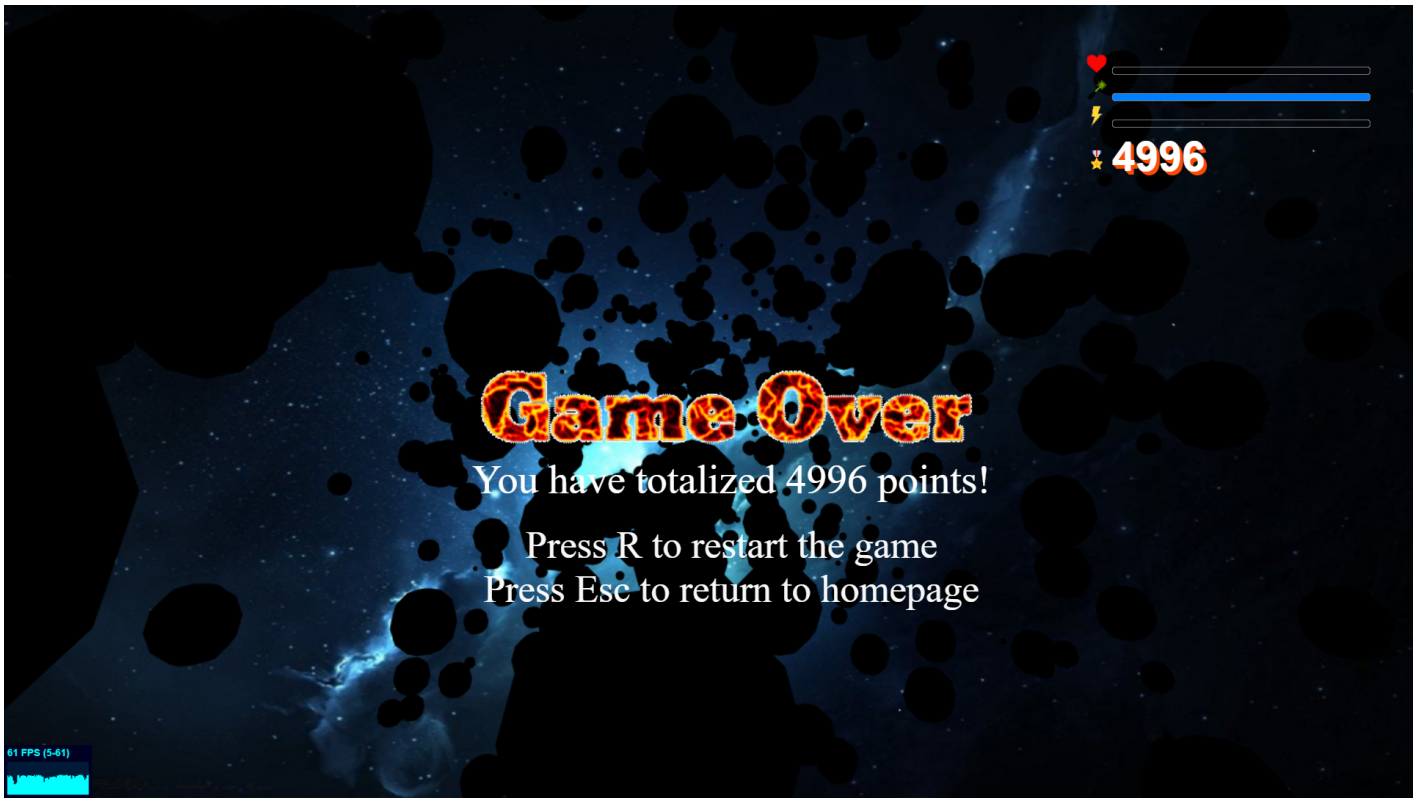
4 Screenshots

4.1 Web site



4.2 Game


4278
60 FPS (7-60)


4996
Game Over
You have totalized 4996 points!
Press R to restart the game
Press Esc to return to homepage
61 FPS (5-61)