Documentation: Asteroids

Frontera Salvatore 1710456

Introduction

The project proposed is an endless game in which the user have to avoid or destroy obstacles to survive as much as possible. It is based on the rappresentation of a planet, where are placed many asteroids as obstacles, and of a spaceship as vehicle. The game can be divided in two main execution:

- First we have an initial page ("init.html" and "init.js") in which is presented the game and few instructions. The user can choose his prefered option, such as textures and difficulty.
- Second is loaded the real game ("game.html" and "game.js"), where the parameters selected by the user on the first part are setted, and is executed the main instruction flow for the execution of the program.

The user interacts with the game controlling the spaceship with the usage of the keyboard (WASD key or arrow keys for the movement and key space for shooting). To verify the result of a match is setted also a score counter so that the gamer can compare his results.

Environment

The environment used is based on two more advanced libraries than WebGL: ThreeJS and TweenMax. The first is a Javascript library and API that do the usage of WebGL, useful to create and execute models and animation for 3D computer graphics on the browser web. It permits to compute in an easier way 3D rappresentation that requires much effort in other basic environment such as WebGL. On the project it is used as main library, so to implement the main functions (for example rendering, scenes, camera etc.). The second is a popular animation tool contained in GSAP (GreenSock Animation Platform), known for its efficiency on loading. It is useful also to staggered

and concatenated animation on multiple objects. In this case is used only for a small part of the project, in particular to trigger the asteroids when the spaceship hits them with the bullets (used in particular for a feedback to the user), and for the animation of the spaceship during its destruction. It is also used Bootstrap, toolkit for HTML,JS and CSS development, to give a better visual of the buttons, defined for starting the game and for reloading the pages.

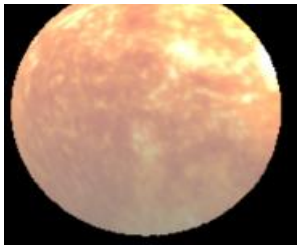
Technical Aspects

Init.html

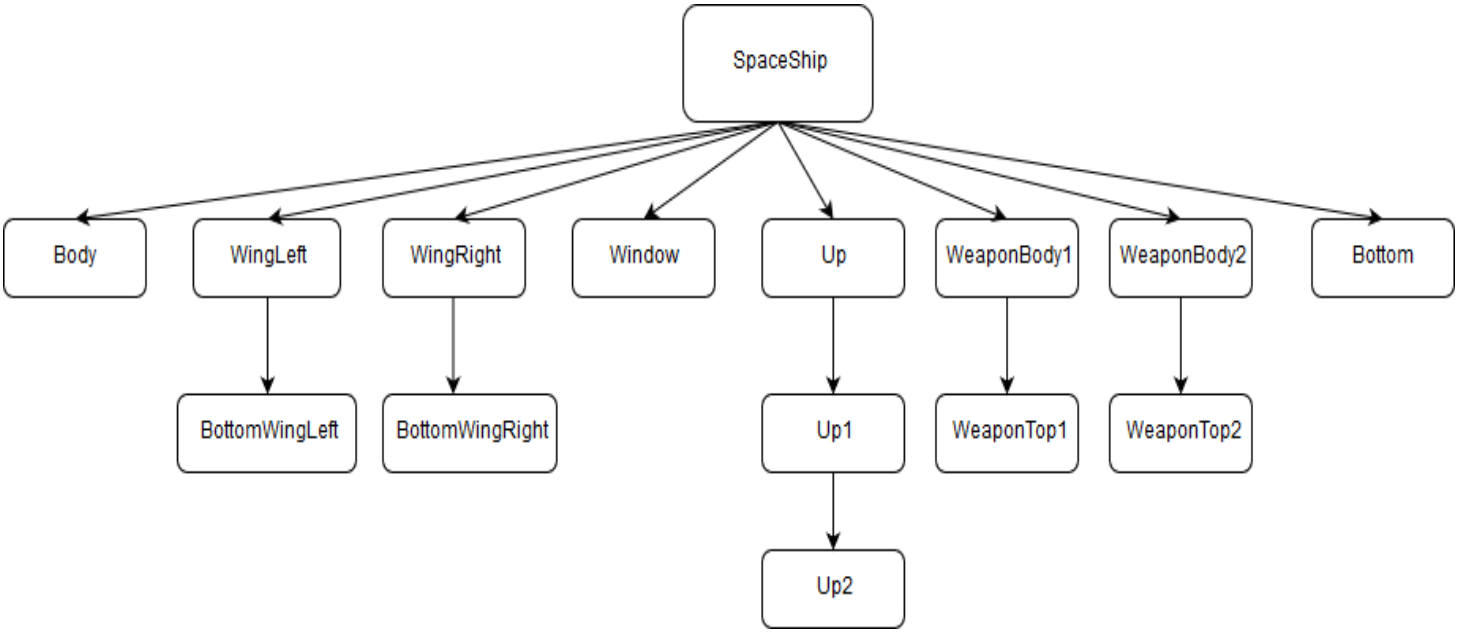
The project starts with the execution of the init.html file. As described before, it rappresents a general idea of the game in which is defined the main goal and the user can decide skins and difficulty of his run. The main work is done on the js file in which is computed the rendering and animation of the initial page, defined on the html file by the tag canvas. In addition to this, there is the main rappresentation of the user interaction, described by the tags “input” of type “radio”, where there is the possibility to choose three types of planets (Earth, Saturn,Mars), three difficulties (easy, normal , hard) and two skins of the spaceship (normal and military). Finally there is the rappresentation of the button “Start Game” where is called the function “executeGame”, described on the header of the file, in which are saved on the url link to the game the value selected by the user. With this method the “game.js” file can get the value from another page and correctly selects the personalization.

Init.js

On the init.js file there is the execution of a simple scene in perspective camera in which are described three main objects: sun, planet and spaceship. The sun, obtained thanks to the class "SphereGeometry" as geometry and "MeshLambertMaterial" as material (to have a non-shiny surface), is useful for having a more descriptive scene and in particular for the lighting, that is combined with it (using "PointLight" function to have an emission of lights in all directions). It is possible thanks to "add" method, that permits to an object to add objects as children, so that is generated an hierarchical model.



As requested by the requirements, the complex hierarchical model is defined by the spaceship. It is defined as follow:



For that is used different type of geometry, such as cylinder (given by "CylinderGeometry" class, in which is possible to handle the radius at the top and at the bottom and the number of faces) and sphere for the window. This model is also used on the

game.js file, in which is only added points at the bottom side for the rappresentation of the propeller during the trip.

Finally there is also a raffiguration of the planet, defined similar to the sun model but with different material and dimension (in particular on the Earth case where are used three different textures), and other lights defined with “DirectionalLight” method (light source that emitts on a specific direction) at the right side and “HemisphereLight” (positioned above the scene).

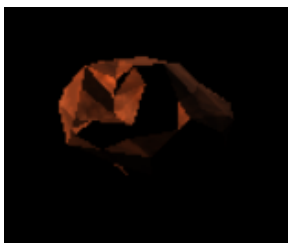
From animation loop point of view is called the function “update” in which is executed the rotation of the described planet and of the spaceship. Moreover are colled “updatePlanet” and “updateSpaceShip” methods to change dynamically the textures of them (for each time a button is checked).

game.html and game.js

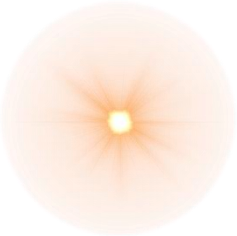
On the game.html file there is a similar behaviour of the init.html file. The main difference is defined by the buttons on the div tag setted as hidden and then, after the end of the run of the game, to visible by js file. They are useful for reloading the page and to return on the init.html page if the user wants to change something.

The game.js file is the main file in which is computed all the game presented. The model previusly described are the same, but there are further objects that we have to take in consideration. Indeed on the behind scene we can see stars defined on the “addStars” function, in which is defined a set of points situated randomicaly on the scene through “Points” method, which is a class to display points that combines the geometry with the material (defined by “PointsMaterial” standard).

Another important model to be considered are the asteroids, that act as obstacle for the spaceship. In this case we can find two different type of implementation. The first is used mainly for the scene environment. In fact on the function “createScene” is called a method “createAsteroids” in which they are setted to high distances all around the planet, so that we have a movement of them during the planet rotation. The second are setted near the surface of the planet with “addWorldAsteroids”



function, called inside each method that defines the planet ("addWorld", "addPlanetMars" and "addPlanetSaturn"). This permit to follow a path defined by an interval of angles in which the rock modelled is situated in a randomic position for each run. Each asteroid is implemented by "createRock" function, with "DodecahedronGeometry" as geometry class (with a randomic position of each vertex) and "MeshStandardMaterial" standard with flat shading (the lights are computed with respect to the key vertex of the triangle and the entire triangle lighting setted as its value computed).



There are three other models not considered yet: explosion, propeller and bullets. The first and the second are rappresented like a geometry of points with a particular texture (for the explosion is used the jpg on the left). They differs for that and also for a different animation (described later). The last one instead is define as a set of sphere geometry without texture, that have a limit lifetime.

The execution of the program can be described as follow. First of all is called the function "createScene", that sets the main configuration of the scene and recalls the models defined previously (planet,stars,lights,sun,asteroids, explosion and spaceship). It is based on a perspective camera, so that we have an idea of depth with respect to the arriving asteroids. It is also enable the shadowMap to compute and render the shadows on the game. The type used is PCFSoftShadowMap, that do the usage of PCSS algorithm (Percentage Closer Soft Shadows). The createScene method verifies the parameter selected by the user, parsing the url of game.html file and selects the right personalization. It calls also the function "addHTML", that adds on the window the visualization of the score and of the energy and power bar in real time.

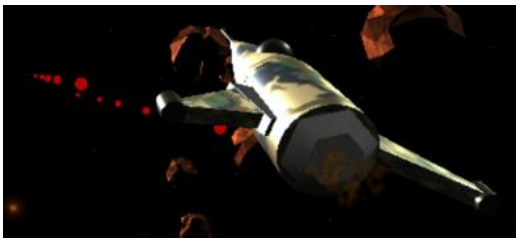
After the execution of this first function the main call is "update" method, that rappresents as init.js the loop of the program. It defines the animation done by the game and other methods to handle the collision of the asteroids with respect to the spaceship

and bullets. The rotation of the planet, together with the life of the spaceship, differs depending on the difficulty entered by the user.

Animation

The animations for this game are different. There are basic animations, such as the rotation of the planet and the asteroids, useful to give a better immersivity to the user and for having an idea of movement of the spaceship with respect to the planet and asteroids. Others are defined for the propeller execution and explosion. As descripted before, they use a similar model but their execution and texture are different. For the propeller is called “propellerExecution” method, in which the vertices defined on the model randomicaly changes their position in a limited area. For the explosion, useful both for the asteroids and the spaceship, is called “doExplosion” method, where, in combination with “explode” method, is setted the position of the explosion of the model and spreads each vertex with respect to a scalar product with the vectors.

There are other animation done with the usage of TweenMax library. These are mainly two: the fall of the spaceship and the trigger of the asteroids.
The first is executed when the spaceship energy reach the 0 value. It is obtained when the spaceship collided to much times with the asteroids (the number of times depends on the difficulty selected). In this case the spaceship is considered destroyed, so starts the execution of the “to” function (from TweenMax library), that animates the object until is reached a destination. It requires the object to be animated, the destination and the duration in seconds of the animation. In our case the spaceship requires two function: one for the position (computed randomly) and one for the rotation (requiring two complete rotation). The function at the end of the execution call another method, in which the spaceship is removed from the scene, does the explosion and ends the animation of the scene).
The second is simpler. In this case is required only a randomic rotation of the asteroids, when a bullet collides them. It is useful in particular to have a feedback with respect to the shooting of the spaceship. Of course each asteroids have a life threshold, so that when a certain number of bullet collides them start the explosion animation and their elimination from the scene.



Finally another type of animation is defined by the movement of the spaceship. In this case is called “keyup” and “keydown” methods, each time there is the press or the release of the keyboard keys. These methods allow the animation of the bullets with respect to the axis, saving them to an array and modifying their position each time, and the rotation and translation of the spaceship (with the usage of the linear interpolation), setting some flag that changes the spaceship behaviour inside the loop method (with the usage of “updateSpaceshipMovement” function).

Collision

From the collision point of view, we can see two different type of function (called both inside update function): “asteroidsLogic” and “bulletCollision”.
The first is used to determine a collision between asteroids and spaceship. In this case there is a checking for each asteroids if the distance between it and the spaceship is less than a value. If it is verified, the asteroid is removed from the array and from the scene, doing also the explosion animation, and is reduced the spaceship life of a certain value. Moreover there is a slowdown of the movements, so that the user can have some instants to handle his situation in a right way.When is reached the zero value, the function starts also the spaceship animation for the explosion.
The second is used to verify a collision between a bullet and an asteroid. For that is required much computational effort. Indeed there is the checking for each asteroid of collision for each bullet. Similar to before, is verified a distance between them and in that case execute the animation described and the removal of the collided bullet. In the case in which the life of the asteroid is equal to zero, is done the explosion and the removal of the asteroid.

Command manual

Here the commands to handle the spaceship:

- A or left arrow key to move on the left
- D or right arrow key to move on the right
- S or down arrow key to move down
- W or up arrow key to move up
- Space bar to shoot

The user can verify in real time its score and the remained spaceship life. Moreover the is rappresented a power bar for the shooting that limits the bullets generated by the user (blocking the shooting events in the case of saturation until the power value does not return to zero).

References

- Some useful hints to implement spaceShip model: https://blog.fps.hu/irjunk-jatekot-three-js/
- Some useful hints for the implementation of the asteroids and stars models: https://codepen.io/Divyz/pen/VPrZMy
- Some useful hints for the implementation of the Earth planet (with combination of textures): http://learningthreejs.com/blog/2013/09/16/how-to-make-the-earth-in-webgl/
- Useful guide for the realization of the asteroids path, modified for my goal, and to give me an idea to create the explosion effect: https://gamedevelopment.tutsplus.com/tutorials/creating-a-simple-3d-endless-runner-game-using-three-js--cms-29157
- Documentations: https://threejs.org/ and https://greensock.com/tweenmax