Interactive Graphics
Final Project

Aloisi Lorenzo - 1836344
Di Tommaso Andrea - 1801802

Contents

List of Figures

1 Environment

1.1 three.js

three.js is an easy to use, lightweight, cross-browser, general purpose 3D library based on **WebGL**.

To be able to display anything with three.js, we need three things: **scene, camera and renderer**, so that we can render the scene with camera.
There are a few different cameras in three.js, we used a **Perspective Camera**.

1.2 WebGL

WebGL (short for Web Graphics Library) is a JavaScript API for rendering interactive **2D and 3D graphics** within any compatible web browser without the use of plug-ins.

WebGL is fully integrated with other web standards, allowing **GPU-accelerated** usage of physics and image processing and effects as part of the web page canvas. WebGL elements can be mixed with other HTML elements and composited with other parts of the page or page background.

2 Credits

2.1 Libraries

2.1.1 tween.js

The tween.js library has been used to realize **smooth animations**, indeed a tween is a concept that allows you to change the values of the properties of an object in a smooth way.

This library allows you to modify the **position** or the **rotation** of an object and there are some functions that let to build a very original animation like: "repeat" e.g. repeat(10) 10 times after the first tween and stops or repeat(Infinity) repeats forever, "chain" i.e. setup one tween to start once a previous one has finished, "onComplete" i.e. function executed when a tween is finished normally.

The most important feature of this library is the **interpolation function**: tween.js will perform the interpolation between values (i.e. the easing) in a linear manner by default, so the change will be directly proportional to the elapsed time.
This behaviour can be changed, for example with "Quadratic.In" the tween will result with a slowly starting to change towards the final value, accelerating towards the middle, and then quickly reaching its final value; instead of the "Quadratic.Out" that would start changing quickly towards the value, but then

slow down as it approaches the final value.

These are all the **interpolation function**:

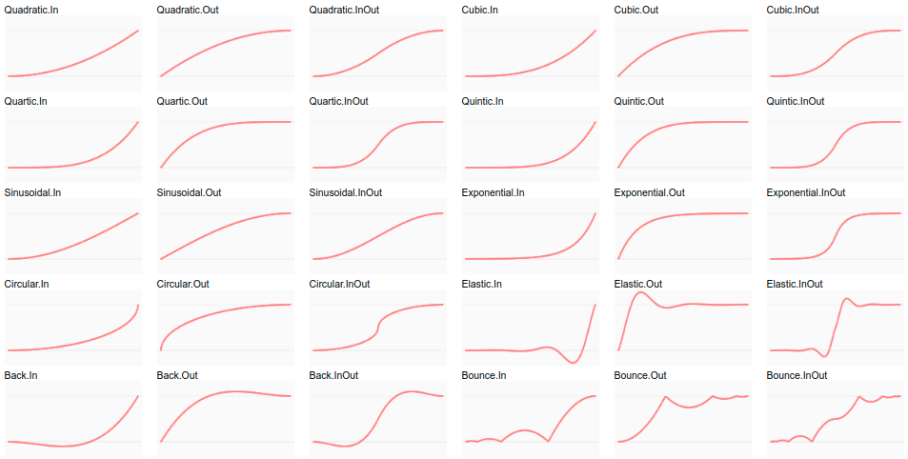


Figure 1: Interpolation functions

2.1.2 jQuery

jQuery is a fast, small, and feature-rich **JavaScript library**. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

We chose to use jQuery since it greatly **simplifies** JavaScript programming, by making it easier to traverse and manipulate HTML documents.

2.2 Additional tools

2.2.1 Bootstrap

Bootstrap is the most popular HTML, CSS, and JavaScript **framework** for developing responsive, mobile first projects on the web.

It includes HTML and CSS based **design templates** for creating common user interface components like forms, buttons, navigations, dropdowns, alerts, modals, tabs, accordions, carousels, tooltips, and so on.

Bootstrap gives you the ability to create flexible and responsive web layouts with **much less effort** and that's why we chose to include Bootstrap in our project: we have been able to save a lot of time and effort with it.

2.3 Models

Every gltf model we used has been taken on Sketchfab.

2.3.1 Road

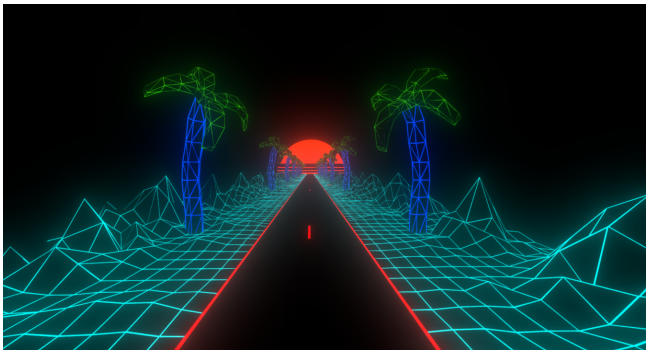


Figure 2: Road

We chose to use this model to better convey the 80s-lofi atmosphere.

2.3.2 Car

Figure 3: Car

This model has a hierarchical structure (3.3.1), which we animated manually (3.3.2)

2.3.3 Coins

Figure 4: Coin

This is the model we used to implement the player's score functionality by randomly spawning sets of coins (3.2.1). We animated it manually (3.4.3).

2.3.4 Obstacles

Figure 5: Obstacle

This is the model we used to implement the obstacles functionality. Just like the coins, the obstacles will be randomly spawned (3.2.3).

2.3.5 Nitro tanks

Figure 6: Nitro tank

This is the model we used to implement the car's invincibility functionality. Getting a nitrogen tank will trigger the car's invincibility. Just like the coins and the obstacles, these nitro tanks will be randomly spawned (3.2.2).

2.3.6 Invincibility bubble

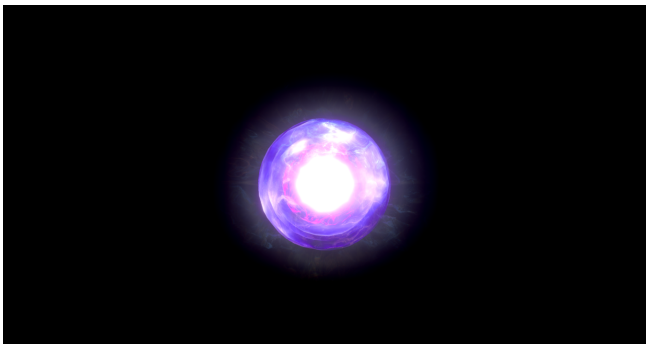


Figure 7: Invincibility bubble

This is the other model we used to implement the car's invincibility. After getting a nitrogen tank, an invincibility bubble will spawn around the car, following it (3.2.2).

3 Technical aspects

3.1 Browser compatibility

Browser	Issues
Chrome	None
Firefox	Audio
Edge	None
Safari	Slow processing

3.2 Items spawn

While the game goes on, some items will **spawn** on the road, to facilitate the car or to get in its way:

- **Coins**: these give 10 points to the user when taken.
- **Nitro tanks**: these grant invulnerability to the user when taken.
- **Obstacles**: the user will lose a life every time the car collides with an obstacle.

3.2.1 Coins

Coins have two main ways to spawn: in a *straight line*, or in a *curve* over an obstacle.
Coin groups will spawn with a fixed distance of 3 along the z axis, between each group.
The function ***random_bitcoin_spawn()*** works as it follows:

- to keep the scene fresh and clean, we decided to adopt a **maximum distance** from the camera, within which the coins will keep spawning;
- the function will **check** if the coins are going to spawn over an obstacle: if true, the coins will spawn in a curve over the obstacle in group of 5; if false, the coins will spawn in a straight line in group of 3. The spawn will of course happen on a random lane of the road, by randomizing the x position of the coin groups;
- after the spawn, we decided to implement also a ***bitcoin_free()*** function, to keep the scene fresh and clean. This function will **remove** from the scene every bitcoin that has already been surpassed by our camera

We clearly also implemented a ***bitcoin_collision()*** function which will check if the car has collided with a coin, updating the user's score and removing the taken coin from the scene.

3.2.2 Nitro tanks

As we said before, once taken, a Nitro tank will grant invulnerability to the user's car for 3 seconds. Just like for the coins, we implemented a function that randomly **spawns** Nitro tanks over the road, with a distance of 100 along the z axis between each tank.

random_nitro_spawn() will spawn a Nitro tank on a random lane of the road, by randomizing its x position. The function will **check** if the tank is going to be spawned over an obstacle: if true, the tank's z position will be incremented by one; if false, the tanks will be spawned with no issues.
After the spawn, a ***nitro_free()*** function will delete from the scene every Nitro tanks that has already been surpassed by the camera, to keep the scene clean.

Clearly we also implemented a ***nitro_collision()*** function which will check if the car has collided with a tank, and will manage its **invulnerability** by spawning a bubble around the car itself.

3.2.3 Obstacles

As we said before, if the car collides with an obstacle, the user will loose a life. Just like before, we implemented a function that spawns obstacles randomly. Every obstacle takes up **two lanes** and will spawn with a fixed distance of 15 along the z axis between each obstacle.

random_obs_spawn() will spawn an obstacle on two random, clearly consecutive, lanes.

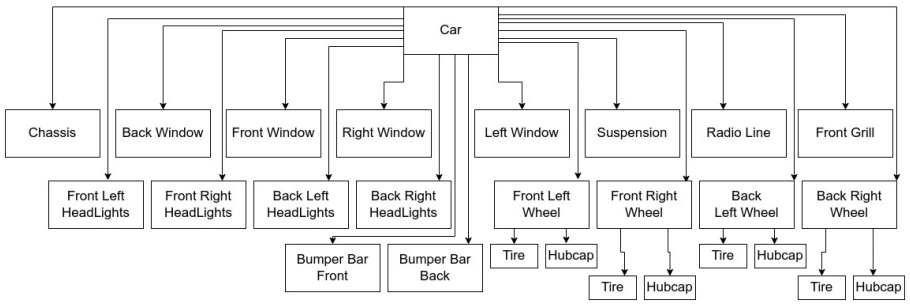
obs_collision() will check if a collision has occured, in which case, the lives count will be decremented by one.

blink() instead will make the car blink: once a collision has occured, the car will start to disappear and reappear really fast, to simulate a blink effect.

3.3 Car

3.3.1 Hierarchical model

The car is loaded through a "gltf" model and thanks to his hierarchical structure we had the possibility to select each part of the model.
This is the representation of the **hierarchical model**:



3.3.2 Animations

The car is the principal object of the game and indeed it has the most accurate **animations**.
In the game the car is always in movement and hence we realized the **continuous rotation of the wheels**:
we built an axis that starts from the center of the front left wheel and ends to the center of the front right wheel; in this way we could rotate the front wheels in radians endlessly to obtain an excellent effect of a car traveling on the road. The same thing is done on the back wheels, obviously building another axis.
To highlight this animation, we applied a **rainbow texture** on the wheels: taken each wheel thanks to the hierarchical model, we mapped the texture and the effect is very nice like a spinning pinwheel.

The goal of the game is avoid the obstacles and take the coins with the car and hence we added the animation to **turn left**, **turn right** and to **jump**.
First of all we assigned to the button "A" the turn left, to the button "D" the turn right and to the button "spacebar" the jump movement.
These animation are all done with the tween.js library: to **turn left** the car, we created a tween that modify the position of the car (take with the hierarchical model) on the x axis and the rotation of the car on the z axis with a "Sinusoidal.Out" interpolation to make the animation more realistic; contemporary with another tween we modify the rotation of the front wheels (take with the hierarchical model) on the z axis to simulate the rotation of the steering wheel and to make the turn left animation more realistic.

The same animation is realized for the **turn right**, obviously with different direction.

For the **jump** of the car, we built two chain of tweens where the first chain modify the position of the car on the y axis and the second chain modify the rotation of the car on the x axis, either with a "Quadratic.Out" first and a "Quadratic.In" after interpolation: in this way we have a smooth jump of the car that wheelies while is jumping.
We put the possibility to turn left or turn right together with the jump to have more fun in the game.

Another animation starts when you **lose the game**, indeed since you haven't your lives anymore, we thought to an **explosion** of the car: with the function "traverse" of three.js we take each child of the hierarchical model of the car and create a tween that modifies each position on all the three axis with the "Exponential.Out" interpolation. In this way we obtain that at each game over, each part of the car is shot in a random direction.

The last animation instead, is when you **win the game** and we made the car **drift**: first we have a tween that modify the rotation of the car on the z axis for the drifting of the car and then we created a chain of tween that modify the rotation of the car on the x axis to wheelie the car. In this way we have a realistic animation of the car stop.

3.4 Other animations

3.4.1 Camera

We realized the animation of the **camera**, indeed we set the initial position of the camera in a point where it frames the road from above in the loading screen and when the loading is finished the animation starts: we created a tween for the translation of the camera on all three axis that reaches the car from behind and contemporary with another tween we modify the rotation of the camera that first frames Mars and then frames the Sun.

Always on the camera, we assigned to the **arrow** of the keyboard the animation to move the camera: we created two tweens to change the position of the camera on the y axis for the up and down arrows and another two tweens on the x axis for the left and right arrows.

3.4.2 Coin attraction

When the player takes the nitro, the car **attracts** the coins around and hence we created an animation where the coins go towards the car: for each coin we built a tween to modify the position on all three axis to reach the position of the car.

3.4.3 Coins

We added an animation also on all coins: they **rotate** around themselves on all three axis.

3.4.4 Mars

For Mars we added a translation on the z axis so that it travels together with the car and a rotation around itself like a real planet.
Obviously also the car and the camera are translated on the z axis for the goal of the game.

3.5 Audio

The background audio and the audio effects are all loaded in the play.html. The background audio starts when the page play.html is loaded, instead the audio effects start when for example there is a **collision** with a coin, with the nitro or with an obstacles, or there is an event like the **jump** of the car and then there is an audio effect for the **game over** and for the **win** of the game.
There is the possibility to **mute** all the audio through the sound icon.

3.6 Scenario

3.6.1 Lights

There is a total of two lights in the scene:

- A pink (#FFDBDB) color **ambient light**, with a high intensity.
- A red (#FF0000) color **directional light** with a high intensity, positioned right inside the red sun at the end of the road and pointed towards the start of the road.

3.6.2 Mars

In the background there is Mars, it walks alongside the car. Mars is a sphere with **textures** of different kinds: a *color*, a *normal* and a *specular*. The animation and the textures make the sphere look like a real planet.

3.7 Post-processing

Before the start of the game, is possible to select two options of post-processing: the **bloom** and/or the **blur**.
We created an **effect composer** where we added the render pass that is created with the scene and the camera.
Then if the bloom is selected, we create an **unreal bloom pass** with the width and the height of the window and add it to the composer; if the blur is selected we create and add two shader pass, one **horizontal blur shader** and one **vertical blur shader**, to the composer.

4 Interactions

4.1 Index page

In the index page we chose to implement the possibility for the user to change difficulties and to select the post-processing effect (blur and/or bloom).
The index page also contains basic information on **how to play** the game.

4.1.1 Difficulty choice

The user can choose between **three difficulties**: easy, normal and extreme.
The choice of each of the three difficulties will affect the car's speed and the number of lives the user will have.

Clicking on the left arrow will trigger the ***decreaseDifficulty()*** function, which will indeed decrease the difficulty.

A click on the right arrow will instead trigger the ***increaseDifficulty()*** function, which will of course increase the difficulty.

4.1.2 Post-processing choices

A click on the checkboxes from this section will trigger respectively the ***changeBloom()*** or the ***changeBlur()*** functions.

4.1.3 Play the game

Clicking on the big **Play** button will redirect the user to the main game page (*play.html*).
A click on the Play button will also pass all the choices we've done in the index page to the *play.html* page by exploiting the **Url Parameters** functionality.

4.2 Game

4.2.1 Commands

The commands of the game are:

Key	Event
A	Turn left
D	Turn right
Spacebar	Jump
↑	Camera up
↓	Camera down
→	Camera right
←	Camera left

4.2.2 Back to the index page

Once the user will have exhausted all the lives, or will have reached the end of the game, a **Game Over** or a **You Win** will appear. In both the cases, under them in the page will also appear a **Go Back** button, which will redirect the user to the index page, making it easier to play again the game.

4.2.3 Mute and unmute

In the top left of the page there's also a button that allows the player to manage the game's audio.
Clicking on it a first time will trigger a ***mute()*** function, which will mute every audio source. Clicking on it a second time will instead trigger a ***unMute()*** function, which will unmute every previously muted audio source .