SAPIENZA
UNIVERSITÀ DI ROMA

Super Mario Runner

Interactive Graphics Project

Faculty of Information Engineering, Informatics, and Statistics
Master Degree in Engineering in Computer Science

Giuseppe Prisco
Matricola 1895709

Professor:

Marco Schaerf

A.Y. 2022-2023

Index

1. Introduction

1.1 Project Idea

The main idea for the project consists of an “Endless Runner” game, a genre of games in which the player character runs for an unlimited amount of time while avoiding obstacles and collecting items along the road.

The game I developed, called “Super Mario Runner”, was inspired from the popular game “Subway Surfers”, in which there is a character running on binaries who can switch between three possible lanes. Other popular games from this genre include “Sonic Dash” or “Temple Run”.

The main reason I chose to develop a game belonging to this genre is that in the past I have already tried creating an Endless Runner game in “Unreal Engine”, but due to the small amount of time at disposal, the project ended up being abandoned.

1.2 Libraries Used

1.2.1 THREE.js

The main library employed in the project is THREE.js, a lightweight, cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL.
It allows you to easily manipulate everything within the scene, including cameras, lights and shadows, fog, 3D objects and geometries, and much more.

1.2.2 Tween.js

The second library I used for the project’s development is Tween.js, an open source Javascript tweening engine for creating simple programmatic animations.
It greatly facilitates the selection and manipulation of a 3D model’s joints, allowing to apply different transformations to animate objects with the use of custom easing functions, for example linear, quadratic or exponential functions.

1.3 Browser Compatibility

The game was tested on the most common browsers, to estimate and assess how good is the performance obtained, in the form of fps (frames per second).
The best performances were obtained in Google Chrome and Microsoft Edge.

2 Environment

In this section I present the environment of the game, consisting of menus, maps and 3D models included in it.
Additionally, several other aspects are introduced, ranging from the gameplay and commands of the game, to the texture, light, shadow, fog and audio components employed in the project.

2.1 Menus

The game allows you to navigate and select different options through the use of menus. They consist of the “Main menu”, the “Settings menu”, the” Pause menu” and the “Game Over menu”.

2.1.1 Main menu

The Main menu is the one that appears after the game is loaded. It presents the different options the user can select before entering the game itself.

Starting from the top, the different options are:

- **Select a playable character**: the user can select the character to play with in the game between “Mario” or “Luigi”
- **Choose a difficulty**: the user can select the different difficulties for the game, which consist of “Easy”, “Normal”, “Hard” or “Godmode” difficulty
- **Choose an environment**: the user can choose in which environment to play the game, among which there are the “Grassland”, “Cave” and “Space” environments
- **Start Game**: by clicking this button, the game starts with the chosen options

2.1.2 Settings menu

This menu can be accessed by clicking on the gear icon in the top-right corner of the home page of the game.

Differently from the Main menu, this menu allows to choose more advanced options, including:

- **Turn the music On/Off**: this button allows to toggle the music present in the home page and in the game
- **Turn the sound On/Off**: this button allows to toggle the sound present in the home page and in the game
- **Choose the quality**: this option allows to choose between three different graphic qualities for the game, including a “Low”, “Medium” and “High” graphic setting
- **Dev options**: by clicking this button, additional objects will be visible in the home page as well as in the game, some of which consisting of axis and grid helpers, camera, light and shadow helpers

2.1.3 Pause menu

During the game, by either clicking the “Escape” or “Space” character on the keyboard, the Pause menu will appear at the center of the screen.

This simple menu contains buttons to toggle the audio and music settings, a button to exit the game and a button to resume the game.

2.1.3 Game Over menu

When the playable character reaches 0 lives, the Game Over menu will appear. Similarly to the Pause menu, the Game Over menu contains buttons to toggle the audio settings, a button to return to the home page and another one to restart the game.

2.2 Map

The map is where the game is set. It is divided into two main sections consisting of the “Main road”, where the character can move, and the “Sides of the road”, which contains non-interactive objects placed outside the main road.

2.2.1 Main Road

The main road is placed at the center of the map and consists of three lanes in which the playable character can freely move.
Additionally, other objects are placed on this part of the map, consisting of different hazards that will “hurt” the playable character and power-ups which will have different effects on the character.

2.2.2 Sides of the road

This portion of the map consists of the two sides of the non-playable part of the game, placed at the left and right of the main road.
In this section of the map different environmental objects are randomly placed, depending on the “Environment” setting chosen before starting the game.

2.3 Models

For the project I used many 3D models, all of them downloaded from the “Sketchfab” website and in GLTF format. They consist of the playable characters, the hazards and power-ups appearing on the main road and other objects appearing on the sides of the road.
All the models used in the game do not contain any imported animation and are directly animated in javascript, through the use of the Tween.js library mentioned at the beginning.

2.3.1 Playable Characters

The playable characters are by far the most complex used in the project. The two characters to play with are “Mario” and “Luigi”, both from the “Super Mario” franchise.

2.3.1.1 Mario

Mario is the default character the user can play with and is a hierarchical model consisting of 66 different bones.
For the purpose of the project, only a subset of these bones has been referenced to later be manipulated during all of Mario’s animations and consists of 49 bones.

2.3.1.2 Luigi

Luigi is the second of the two playable characters and, similarly to Mario, is a complex hierarchical model. Luigi is composed of a different set of 69 total bones and also for him only a subset of them has been manually referenced, consisting of a group of 49 bones.

2.3.2 Hazards

The hazards are 3D objects that will cause the playable character to lose a life if it collides with them. Since the playable character can only move on the main road, these objects are randomly spawned on tiles corresponding to the main road.

2.3.2.1 Spikes

The spike is a simple object placed on the lower part of the track. It can spawn in any combination of two or three spike objects in a specific row of the track.

2.3.2.2 Rollers

The roller is a simple object that is placed either vertically (covering both the lower and upper part of the track) or horizontally (covering the upper section of the track). As with the spikes, different combinations of rollers can spawn on the track, with the top ones consisting of a group of two rollers merged together.

2.3.3 Power-ups

The power-ups are simple 3D objects placed on the main road that can be interacted with the playable character. They have different effects on the player when it collides with them.

2.3.3.1 Coin

When collected, the coin allows to increase the score obtained during a run. It can be placed in a series of 2 to 4 coins and can spawn in either the top or bottom side of the road.

2.3.3.2 Mushroom

The mushroom allows the player to increase the number of lives during the run. When at max lives, the collected mushroom will increase the score obtained during the run. This object can spawn in the top as well as the bottom part of the road.

2.3.3.1 Star

When collected, this object allows the player to become “Invincible” for a short period of time. During this period, the player will not take damage when colliding with the other hazards and consequently will not lose any life.

2.3.4 Other Objects

This group of objects consist of all the 3D models that are spawned on the sides of the road. These objects are not animated and their purpose is to fill the surrounding environment and distinguish between one environment and the other ones.
These objects include random elements taken from the Super Mario franchise, or other more general elements such as the cave sides, the forest or the space elements.

2.4 Gameplay

The purpose of the game is to get the highest score possible while avoiding taking damage.
As a reminder, the game is not meant to be balanced (with respect to the difficulty, number of coins or powerups spawned ecc.) to also provide the user the possibility to explore every aspect of the game, options and features.

2.4.1 Commands

Once playing the game, the playable character can be controlled with the following inputs:

- "**A**" or "←": move left
- "**D**" or "→": move right
- "**W**" or "↑": jump
- "**S**" or "↓": slide
- "**Space**" or "**Escape**": pause and resume game

2.5 Scene

In the project there are mainly two "scene" objects:

- the first one belongs to the home page, in which the playable character is displayed
- the second one belongs to the game itself, with all the tiles and objects generated in a run

The two scenes are characterized by different properties and objects, which are analyzed in the following sections.

2.6 Textures

Most of the 3D objects already have different kinds of textures characterizing them, in particular they fall in the following categories:

- diffuse texture
- normal texture
- specular texture
- emissive texture
- metallic texture
- roughness texture

In addition, the tiles composing the main road and the sides of the road are assigned different color textures based on the environment chosen in the settings, before starting the game. These color textures have been downloaded online and slightly modified by me to better match the style of the game.

2.7 Lights

For the project a different number of lights has been used, both in the home page where the 3D model of Mario or Luigi appears as well as in the game.

2.7.1 Ambient light

The ambient light is used to illuminate all the objects present in the scene with the same intensity. I choose a white light as the ambient light for both the scene present in the home page and the in-game scene.

2.7.2 Emisphere light

The emisphere light is used in THREE.js to supply an additional light source which fades from the sky color (a pale, faint yellow) to the ground color (a dark, shadowy blue). This light is positioned directly above the scene and is employed in both the home page and the in-game.

2.7.3 Directional lights

A directional light is a light that is emitted in a particular direction and will behave as it is placed infinitely far away. It is used to simulate the light emitted by the sun since the rays produced from it are all parallel.
For the home page scene I used one directional light with the property “castShadow” equal to true so that it will cast the shadows of all the objects illuminated by it.
For the in-game scene I employed a directional light that illuminates the floor and that follows the player, also with the property “castShadow” set to true. Additionally, I used a set of three additional lights parallel to the tiles of the terrain to obtain a better visual effect for the metallic objects present in the scene.

2.8 Shadows

Shadows are directly related to lights: lights that are used to cast shadows have a shadow camera that defines the portion of space in which shadows are casted by the objects.
For the home page I did not need to modify the default values for the shadow camera to obtain a good effect while for the in-game scene I tuned the parameters to make the shadow camera follow the player during the run and correctly display the shadows for all the run’s duration.

Moreover, since the shadows are computed from a directional light (which by definition produces parallel rays of light), an orthographic camera is used to compute shadows instead of a perspective camera.

2.9 Fog

THREE.js allows you to implement a fog effect to the scene with the “Fog” constructor: by providing the color, the near and far planes for the shadows I could add fog to the scene.
In particular I decided to add a white fog for the “Grassland” environment, a more dense and dark fog to the “Cave” environment and lastly, a dark-blue fog for the “Space” environment.

2.10 Audio

To provide a more realistic experience for the user as he is playing a real videogame, I decided to add the audio to the game.
By creating an Audio listener and attaching it to the camera, I could define Audio objects created as non-positional audio sources, which represent a global audio object. This was done to make the user perceive the audio in the same way independently from what happens in the scene.

However, there is an additional problem in that some browsers do not allow for audio files to be executed without a previous interaction by the user (for example clicking on a button). To cope with this problem (and to also allow all the assets to be loaded in the browser) I decided that the user could enter the website only by first clicking on a button, which would show up only after all models, textures and sounds have been correctly loaded.

The three audio objects created for the project represent a music channel and two sound channels, respectively. This allowed me to play many audio files at the same time without stopping and resuming the previous ones. The audio files used for the project have been downloaded online, having some of them transformed by me with pitch editing and shifting to obtain a better sound effect for the game.

3 Technical aspects

In this large section of the document I will present more detailed aspects of the game, ranging from loaders used, event listeners, game settings, character and objects initialization, map generation and much more.

3.1 Loaders and Managers

In order to properly load every asset used in the game, the following pattern has been used for loading 3D models, textures and sounds:

- Initialization of a LoadingManager() object
- Initialization of a specific loader to load 3D models (GLTFLoader), textures (TextureLoader) or sounds (AudioLoader)
- For each element of the asset category considered, the element is loaded and then is assigned some properties that will later be used

When entering the website, the first assets loaded will be 3D models, followed by the textures and finally the sounds.
Additionally, during the loading of elements, a progress bar will continuously be updated to show the user the percentage of assets loaded at any time.

3.2 Camera animations

The camera is characterized by 3 different animations:

- The first animation simply places the camera such that both characters are initially visible in the home page scene, and immediately after zooms on the current playable character
- The second animation is performed when selecting a playable character: when choosing a character by clicking on their respective button, the camera will do an half-circle movement to position itself in front of the new character
- The last animation for the camera is when it follows the character during a run of the game: it constantly updates its position so that it is placed directly behind the playable character and a little higher than him, looking at the ground floor at a slight angle

3.3 Responsive Renderer

The renderer has been made responsive such that when the window is resized by the user, the camera aspect ratio and the renderer size is updated accordingly with the current values of the window inner width and height.
Moreover, the camera aspect ratio and renderer size assume different values depending if we are in the home page or in the game: if we are in the home page, the width of the renderer will be equal to half of the size of the screen and the height would be at full screen, on the other hand, if we find ourselves in the game, the renderer size will be equal to the full screen in both width and height.

3.4 Game Settings

The game allows the user to choose between different settings, by clicking on the buttons found in the different menus. When a button is clicked, a global variable, named “settings”, will be updated according to the current setting chosen by the user, in particular the variables will be updated in the following way:

- Choose a character → the **playableCharacter** variable will be updated
- Choose a difficulty → the **difficulty** variable will be updated
- Choose an environment → the **environment** variable will be updated
- Toggle music → the **playMusic** variable will be updated
- Toggle sound → the **playSound** variable will be updated
- Choose the quality → the **quality** variable will be updated
- Dev options → the **dev** variable will be updated

Additionally, a broader set of other variables will consequently be updated when we are in the game, when the score and lives are updated, when the character is invincible ecc.

3.5 Event Listeners

A set of event listeners has been used to make the whole website more interactive: they are related to buttons, keyboard inputs and window resizes.

The buttons have by far the most articulated objects of this section, by having different effects when in a click, over or active states and by also updating the related settings or moving between pages.

When a button is clicked, the corresponding setting will be updated, the current button will assume a different color to highlight that it is currently clicked and, additionally, the other buttons of the same section will be deselected and will no longer be active.
For example, when clicking on one of the difficulty settings, the other ones will be deselected.

When not in the game, all the keyboard inputs are blocked (the input will be shown in the console). Only when playing the game the keyboard inputs will be active, allowing the user to input commands for moving the playable character and pausing the game, as already anticipated in the "Commands" section.

As already said before, an event listener for when the window is resized has been used to correctly update the camera aspect ratio and the renderer size.

3.6 Character Initialization

The playable character needs to be initialized before showing it in the scene to the user. It is first assigned a reference to each bone used for the animations (45 in this case) and later the correct animations could play.

I will go more in-depth in describing all the animations belonging to this project in a later section, thus, here only the type of animation that is currently played will be presented. Generally, before each animation, a characterReset() animation will play in order to place the bones of the character in a default position (it is a basic pose, with the character standing and arms slightly rotated, hands closed).

If we are in the home page, the current playable character will be in his idle animation (basic animation to only show some movement for the character) and a more interactive animation, the characterFollowMouseAnimation() will play. The latter animation consists of updating the head and spine rotation of the character depending on the current position of the mouse, simulating a head-following animation.

If we are in the game, the character will be in his default characterRunAnimation(), in which he moves many bones to simulate that he is running.

3.7 Animations handling

In order to correctly manage the character's animations, a set of global variables has been used, consisting in:

- characterRunningAnimationTweens
- characterCurrentAnimationTweens
- characterMovingAnimationTweens

These 3 variables are lists that contain all the tweens (that are simply tween objects that keep track of the current position during an animation) that are related to a specific character animation.

The characterRunningAnimationTweens list contains all the tweens that are present in the character running animation, the characterCurrentAnimationTweens list contains the tweens of all the current active animations and the characterMovingAnimationTween list is used to keep track of all the objects tweens but also the mario moving forward animation tweens.

These lists are manipulated whenever the game is paused (pausing all the current tweens), resumed (resuming all the tweens), or when the character initiates the jumping or sliding animation (for which only the characterRunningAnimationTweens tweens are paused when the animation starts and resumed when the animation ends).

3.8 Main Game loop

The whole game revolves around one big loop function (in reality, only until the player reaches the end of the game, set very far and thus, in principle, unreachable).

This function, called “moveCharacterForward()”, will cover different aspects of the game, many of which will be treated in a later section (precisely for checking the objects' collisions and removal of tiles). In particular we have:

- The move character forward animation, which simply decrements the z position of the playable character, making him move on the main road backwards on the z-axis (the character is initially rotated to face the negative direction of the z-axis)
- The camera following the player, so that it is always at the same distance from the playable character, at the same angle (the relative position between the camera and the character is constant throughout the game)
- The directional light moving along the track (along with its shadow camera, which cast the shadows of the intercepted objects) so that it is always in front of the character, positioned on the left side of the road
- The constant update of the score and lives at any given point, reflecting how many points the user is currently obtained through the run and the current number of lives remaining

- The check of the objects' collisions, to either damage the playable character if the object is an hazard or to give him a particular effect if it is a power-up
- The removal and generation of additional tiles, in order to keep the game going on forever (until the far end of the game is reached)

3.9 Pause Game

When playing the game, it is possible to click "Escape" or "Space" on the keyboard to pause it (with the pauseCharacterGame() function). In doing so, a set of different functions is called, mainly the pauseAllTweens() function and the pause method on the sounds.

The role of pauseAllTweens() is to take a list of tweens as the argument (the 3 lists previously described) and for each of them, call the pause method. This allows the game to simulate its stopping since everything in the scene will stop moving, until the user decides to continue the game.

The pause method used on the sounds will effectively stop them so that they can be resumed when the game continues. For the music, the volume will be automatically lowered when the game is paused to better fit the functioning of the pause menu.

3.10 Map Generation

To generate the map in which the player will move, a series of different functions are called. In particular we have the following steps:

- Textures settings and starting tiles generation
- Objects initialization
    - Matrix initialization
    - Objects instantiation

3.10.1 Textures settings and starting tiles generation

When generating a tile, a set of different textures is assigned to a material depending on the environment chosen by the player. In particular we have two different textures, for the main road and sides of the road respectively, for each of the three environments.

In addition, at the start of the game, a set of 3 tiles will be generated, the first of which does not contain any hazard or power-up on the main road.

3.10.2 Objects initialization

In order to spawn the objects in the scene, two functions are called:

- initMatrix(), which creates a matrix containing in it the objects' names in specific positions
- initObjects(), which effectively spawn the objects in the environment

3.10.2.1 Matrix initialization

To populate the map with objects, a matrix composed of 18 rows is created: in particular, I decided that row number 1 is devoted to spawn hazards (rollers and spikes) on all the three different lanes. Additionally, there is a small probability of spawning a star in the same row.
The mushrooms can only spawn in row number 5 while two-lane hazards and coins will spawn on row number 9.

Depending on the quality settings and on the environment chosen by the user, a different number of columns and objects will be selected.

To assign a specific object to a position, I simply store in the cell of the matrix mat[i][j] the name of the object to be spawned. They are all randomly selected based on different probabilities such that the new tiles will always be different

This matrix represents the spawn of objects for each tile and as anticipated at the beginning of the document, it is purposely created to not place too many objects and therefore making the game excessively difficult.

3.10.2.2 Object instantiation

To spawn objects (that is, adding them to the scene), a check on each matrix's cell is done and depending on its content (the name of the object to be spawned) a sequence of operations is performed:

- Cloning of the 3D model loaded at the beginning
- Setting the scale, rotation and translation parameters of the objects properly, also depending on the position they have in the matrix
- If the object belongs to the hazards or power-ups, it is also assigned a box collision (which is scaled to be slightly bigger than the object itself) as well as an animation (all the objects on the main road are animated)
- The object is finally added to the scene

In order to not slow down the whole website when instantiating an object, the clone method was used on the 3D models loaded at the beginning instead of continuously loading the same models over and over.

3.11 Tiles generation and removal

To not overload the game, the objects and tiles that are not visible anymore in the camera vision cone should be removed from the scene. This is done when the tile's final position exceeds the playable character z position by a certain amount: the tile is removed from the scene, the inactive collision boxes are removed as well as the tweens of the objects not collided with the player, and finally a new tile is generated, following the principles defined in the previous section.

Since the tweens and collision boxes are saved in different lists, when a tile is removed every element in the list is removed as well (with the splice function present in javascript).
The same reasoning applies to tiles, also belonging to a list.

This cycle is repeated every time the player moves forward and allows the game to be played indefinitely.

3.12 Checking objects' collisions

The function to check if the player collided with some object is continuously called in the main game loop.
For each active collision box present in the scene, the function does the following:

- It extracts the minimum and maximum position values for the vertices along the 3 axes
- It does the same for the collision box of the player
- Then it checks if the positions of the 3 axes overlap each other and, only in the case of a collision happening over all the 3 axes, executes the code for the collision

Depending on the collisioning object, it first removes the collision box from the list of active collisions and then either damages the player (if the object is a hazard) or awards him with a particular effect (if the object is a power-up).

4 Animations

I decided to write this portion of the document as a different section entirely since the animations developed for the project and used in the game are in a large amount.

Each animation present in the project has been created by me directly in javascript (with the aid of the Tween.js library) and no objects have imported animations.

Every object present on the main road has been animated and I further distinguish between a character's animations and other objects' animations in the following sections.

4.1 Playable character's animations

Both Mario and Luigi have different animations based on where we are in the game and what command is issued by the user.

The functions that animate the playable characters are basically done such that every function exploits the structure of the hierarchical model to perform the animation. Moreover, for most of them, there is a specific sound that will play when the animation is performed.

4.1.1 Character follow mouse animation

This animation is performed when we are on the home page, and the currently selected playable character will simulate the following of the cursor controlled by the user.
The function works in the following way:

- The current position of the cursor (the mouse pointer) on the screen is saved
- The rotation degrees of the head and spine joints are computed, based on the current mouse position, the percentage of screen in which the cursor is found, and the maximum degree of rotation of the particular joint
- The joint is rotated by the amount found in the previous step

4.1.2 Character idle animation

This animation is continuously executed when the playable character is in the home page and represents the movements he performs when idle.

The animation involves a basic group of bones, each rotating in little amounts, including:

- Body
- Pelvis
- Arms
- Legs

4.1.3 Move character forward animation

This simple animation consists of moving the body of the playable character in the negative z direction.
In particular it involves only the body of the player and lasts an amount in seconds equal to the end position of the game divided by the character speed (the position is hardcoded in the javascript file while the speed can be chosen by the user when selecting the difficulty).

4.1.4 Character reset animation

More than an animation, this function is used to bring the bones of the playable character in a default position, at specific rotations.

It basically involves all the referenced bones, the most important part of the body being:

- Body
- Spine
- Pelvis
- Head
- Arms
- Hands and Fingers
- Legs and Feet

4.1.5 Character run animation

The main animation when we are in game is the running animation performed by the player. It simulates the character running by alternatively swinging the arms and legs back and forth, while also including a movement for the head.

The main bones involved are:

- Spine
- Head
- Pelvis
- Arms
- Legs
- Thighs
- Calfs
- Body

4.1.6 Character slide animation

This animation is performed when the user presses the down arrow or the “S” input on the keyboard. It lowers and rotates the body of the player, while leaning with his hand on the ground, in order to simulate a slide performed by the character.

The slide animation involves the following bones:

- Spine
- Head
- Arms
- Fingers (upper and lower part, including the thumb)
- Hands
- Body

4.1.7 Character jump animation

This animation is performed when the user presses the up arrow or the “W” input on the keyboard. It elevates the body of the player, rotates the right arm and raises it, while also opening both legs, to simulate a realistic jump for the character.

The jump animation involves the following bones:

- Spine
- Pelvis
- Head

- Cap
- Arms
- Legs
- Body

4.1.8 Move character to a direction

This animation is used when the user wants to move the playable character in a specific lane. It simply translates the x position of the player to match the center of the new lane. Additionally, it consists of two types of movements: a legal movement and an illegal movement.
If the character would move to a correct lane (meaning in either of the 3 lanes, starting from a correct position), the x position of the player would correctly be updated.
However, if the character is about to move to an illegal position (for example the player is in the left lane and wants to go left), a different animation would play which simulates an “homing effect” that attracts the character back on the starting legal lane.

4.1.9 Character damage animation

This animation will play when the player collides with a hazard (spike or roller). The visibility of the character will switch back and forth between true and false, meaning that he will constantly appear and disappear.
This is a commonly used animation in many video games and in my case it has a delay of 200 milliseconds and will repeat for 5 times.

4.1.10 Character star animation

This animation is very similar to the damage animation and will play when the player collides with a star (it is in the category of power-ups). The visibility of the character will switch back and forth between true and false as before but in this case it has a delay of 100 milliseconds and will repeat for 100 times (thus will play for much longer than the damage animation).

4.2 Other objects' animations

These objects are characterized by simpler animations than in the case of the playable characters and consist mainly in applying scaling, rotating and translating transformations on the object.

4.2.1 Coin animation

The coin will rotate on its y axis constantly, starting from 0 degrees to 360 degrees.

4.2.2 Mushroom animation

The green mushroom will constantly move up and down by applying a translating transformation over the y axis.

4.2.3 Star animation

The star will rotate on the y axis back and forth, ranging from angles between -60 to +60.

4.2.4 Spike animation

The spike will have a small scaling transformation, constantly applied on the y axis only.

4.2.5 Vertical roller animation

The vertical roller present in the scene will rotate around its y axis, starting from 0 degrees to 360 degrees.

4.2.6 Horizontal roller animation

The horizontal roller will constantly rotate around its x axis, starting from 0 degrees to 360 degrees.

5 User interactions and effects

In this section I briefly summarize all the interactions present in the project based on an user's action.

Starting from the buttons, all of them have an over and active effect and produce a sound both when the mouse passes over them and when the user clicks on them.
The buttons for changing the difficulty and quality settings have a special "bubble" animation when clicked, while the "Godmode" button in the difficulty settings has a special rainbow shadow when it is not clicked and becomes all animated when is clicked.
The "Start Game" button has a pulse effect similar to the "Go" button present at the start of the website to guide the user into clicking it.
The music and sound buttons, as well as the dev option button, change color when they are inactive.

The main menu in the home page has a special animation that gradually shows the options, making them appear in a sequence starting from the top.
The settings menu, on the other hand, will appear from the right-hand side of the page when the user clicks on the settings button (with the gear icon).

Some portions of text have been assigned a blur effect, before being shown to the user.
When collecting a coin or a life, a special animation will play that adds the coin points or life to the total counter.

When entering the home page and starting a game, a special "transition" animation has been used to simulate the motion of the page.

The loader present when first entering the website has been animated to show three dots bouncing on the ground, with the shadow changing accordingly. Additionally, a loading bar has been implemented to show a visual progress for the assets' loading.

All the animations for the menus, buttons, loader and page transitions have been realized through the use of keyframes, present in the css files.

As a side note, some of the animations for the buttons have been taken from previous projects I worked on in the past, specifically the "Linguaggi e Tecnologie per il Web" course of the Bachelor Degree I pursued.

Other interactions with the game are visible by moving the mouse in the home page (the character follow mouse animation will play) or directly when playing the game, in which the user has the ability to control the playable character by using the commands already showed in the "Commands" section, or to pause and resume the game.

6 File Structure and Organization

In this last section of the document I only schematically present what is contained in each file used in the project.

assets folder:

- 3Dmodels folder - contains all the 3D models used in the project
- images folder - contains all the images and textures used in the website
- sounds folder - contains all the audio files, in the form of musics or sounds

css folder:

- buttons.css - contains the graphics and animations of all the buttons (also using keyframes)
- loader.css - contains the animations at the start of the website for when every asset is loaded
- style.css - contains general graphics and properties of the elements of the website
- transitions.css - contains the transition animation from one “page” of the website to another one

js folder:

- script.js - it is the main javascript file and contains the following elements and functions:
    - canvas
    - settings
    - models
    - textures
    - sounds
    - loaders
    - init
        - set canvas size to half screen
        - audio listener
        - window resize event for responsive canvas
        - init scene
            - axes, grid helpers
            - scene background
            - fog
        - invisible plane which receives shadows
        - init characters in the scene
        - animate (render loop)
        - home page music
    - init event listeners
        - button click and hover

        - select default options
        - all the buttons of the game
            - settings
            - buttons to control the audio settings
            - graphics
            - character selection
            - difficulty
            - environment
            - developer options
            - pause menu
            - game over menu
    - start game
        - resize canvas to be at full screen
        - init scene (as before)
        - init keyboard event listeners to control the playable character
        - init the game with the correct character, camera and scene settings
- animations.js - it is the file that contains all the objects' animations as well as function to manage the tiles generation and removal:
    - initCharacters()
    - initCHaracterGame()
    - setMarioBones()
    - setLuigiBones()
    - characterFoloowMouseAnimation()
    - characterIdleAnimation()
    - moveCharacterForward()
    - characterReset()
    - characterIRunAnimation()
    - characterSlideAnimation()
    - characterJumpAnimation()
    - moveCharacterTo()
    - characterDamageAnimation()
    - characterStarAnimation()
    - initCharacterKeyboardEventListeners()
    - pauseCharacterGame()
    - gameOver()
    - initTile()
    - initObjects()
    - initMatrix()
    - removeTiles()
    - removeInactiveTweens()
    - removeInactiveCollisionBoxes()
    - checkCollisions()
    - coinAnimation()
    - mushroomAnimation()

    - starAnimation()
    - spikeAnimation()
    - rollerVerticalAnimation()
    - rollerHorizontalAnimation()
- utils.js - it mainly contains utility functions which are used by other files, and includes the following features:
    - play music
    - play sound
    - pause, resume and stop all tweens
    - function to convert from degrees to radians and vice versa
    - select a random number from an integer interval

libs folder (it contains the external libraries used in the project):

- three.module.js - it is the THREE library, used to manage many aspects of the project
- tween.esm.js - it is the library used to animate the objects smoothly
- GLTFLoader.js - it is a THREE.js sub-module used to manage the loading of 3D models in GLTF file format

index.html - it is the only html file that contains all the elements that are showed in every page of the website