Report Final Project

Elisa De Bellis, Alessandro Ferrante, Edoardo Oren Alonzi

Contents

Only UP

We chose to implement a a captivating platform-climbing game that features a boy as he jumps on various surfaces to reach higher places.

The aim of the game is to find a way to reach the highest point of the map, represented by the platform eith the black and white finish flag.

1 User Manual

When the user loads the index page, has the possibility to choose between two different characters and the difficulty of the game. After one of the character has been selected, to start the game the player must press any key, at that time start the timer. Once the game has been started, the goal of the player is to jump and move on the various object in order to reach the higher place. In order to move on the obstacles the character can be controlled using these commands:

- "W" to move forward
- "A" to move left
- "S" to move backward
- "D to move right
- "SPACE" to jump
- "SHIFT" to run

If the player chose the difficult "easy" then it can use the two checkpoints in the game. So in whatever time, pressing "L" it can return to the last checkpoint reached. Instead if it chose the difficult "hard", the checkpoints are disabled. The game is over when the character arrive to the higher object.

2 Environment

In this section, an overview of the components that constitute the OnlyUp environment will be provided.

2.1 Map

The comprehensive map is composed of various models and shapes, primarily extending vertically. It can be observed in the figure below.

The Map has a base which is actually a low-poly model of a small city of quadrangular form. It is delimited in its boundaries so that when we try to go out or we fall outside the city's streets, the game brings you back to the starting point, which is the center of the city.

The buildings (one of which we should also be able to climb on during the game) and the baseline represented by the level of the streets, have been also given a physical body (a hitbox that traces their positions and dimensions) using the Cannon.js library.

After implementing the base for our platform, we inserted a series of objects (of different kinds and dimensions) in such a way that they all together created a path that was traversable by the player, jumping from one object onto the next one at each step. Starting from the center of the map, we can climb on the van in front of us in order to reach the first building and then we can proceed all the way to the first checkpoint, which is represented by the second building; from here on, the game starts to become a little bit more difficult as objects are often rotated around their axis in such a way that they will enable the player to reach increasingly higher places and, moreover, there will not be any other checkpoint to re-start from until the end of the game.

Figure 1: Map of the environment

Every object model has been imported and placed into the map's environment so that any jump the player has to do to proceed is indeed feasible (the map has been with respect to the character movements) and so that each of them has the right physics attributes (solid body), obviously based on the dimensions of the model.

As we can also see from the figure, starting from the 'plane' model we have two possible paths through which we can proceed and we can choose to go either way; if we go left, the following objects are not so much inclined but we decided to insert a slightly more difficult step before the end. In fact, the book just before the final platform is continuously moving along his Z axis, so we have to wait and select the right time to jump in order to land perfectly on it and go beyond it. The last platform with the black and white flag represents the end of the map (and therefore, of the game).

2.2 Characters

In the game, you have the option to select from two characters: a firefighter and a warrior girl. Both of them are hierarchical models imported but without any kind of built in animation. The characters' movements involve the manipulation of their constituent bones. The characters are as follows.

Figure 2: Firefighter Character

Figure 3: Warrior Girl Character

2.3 Models

All models (excluding pre-existing animations) have been imported from Sketchfab, a website dedicated to 3D modeling and content sharing. Sketchfab functions as a platform for publishing, sharing, exploring, and trading 3D content. It features a viewer built upon WebGL technologies, enabling the presentation of 3D models directly on the web.
Sketchfab relies on the WebGL JavaScript API to seamlessly render 3D content across all major modern web browsers. Importantly, it operates independently of third-party plugins, ensuring a streamlined and plugin-free 3D experience.
There are numerous models, with the largest one serving as the foundation for the map, which is the city model. As one ascends, each object is treated as a separate model, and it is individually imported, relocated, or rotated within the map.

So, each model has its own properties and, along with they're position inside the map, each one has been provided with a suited hitbox (implemented with the Cannon.js library), so that that the character can land on its surface after each jump but it falls when not inside its region.

Two of them also have been given a specific animation function; the first one concerns the translation along the Z axis (delimited by a minim and a maximum value reachable until changing its direction) of the above mentioned book model, while the second one was implemented to let the final flag continuously move (to better give the idea of being the last step of the game).

Finally, we tried to choose models with more or less the same cartoon-ish style. Here below, there are some samples of the imported models:

Figure 4: Camera Model Example

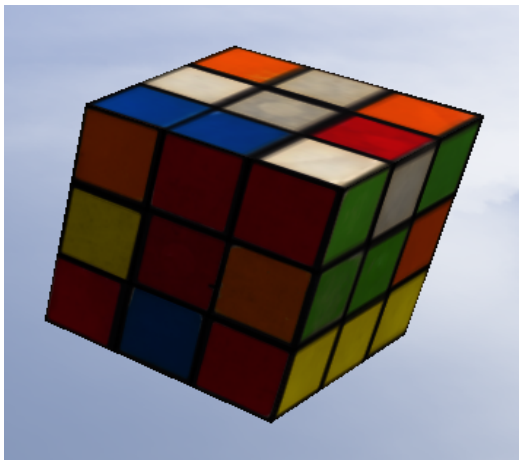

Figure 5: Cube Model Example

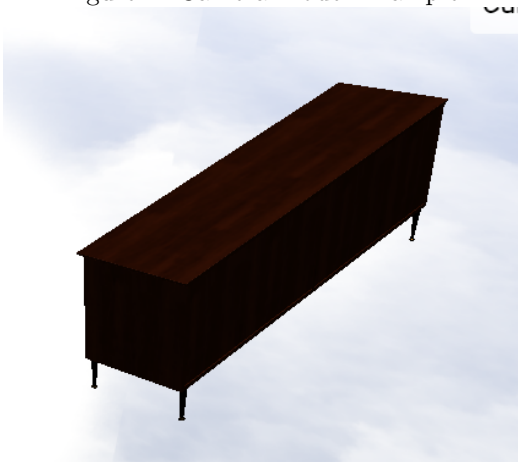

Figure 6: Cabinet Model Example

Figure 7: Table Model Example

3 Technical Aspects

3.1 Character-world interaction management

3.1.1 Finite state machine

The finite state machine is an abstract machine that can exist in a finite number of states at any given moment. In the case of OnlyUp, it was necessary to define a finite state machine for characters in a way that distinguishes animations for each state.
Specifically, in **FiniteStateMachine.js**, the framework for the FSM is created, allowing the creation of states and transitions between them. In **CharacterFSM.js**, the FSM for the in-game character's movement is defined. Four states are specified:

- Idle
- Walk
- Run
- Jump

This division enables the differentiation of animations for each state at every moment.

Within the states defined in this file, all animations are defined from scratch in JavaScript. To access the character's bones efficiently, a dictionary was created to expedite the process of calling each bone in individual animations, resulting in smoother animations. This dictionary was generated from the GLTF files downloaded from Sketchfab, and as a result, no animations were imported. In particular, for all animations, parameters related to the rotation and speed of the arms (divided into 3 parts) and legs (divided into 2 parts), along with hands, feet, and in some cases even the trunk and head, were adjusted.

3.1.2 Character controller

To enable character movement, a **Character Controller** has been implemented. This tool is responsible for detecting user keyboard button presses and releases, subsequently influencing state changes in the FSM and the modification of certain character properties, such as speed. The keys through which the user can alter the states are:

- Forward (W)
- Left (A)
- Right (D)
- Backwards (S)
- Shift (to sprint, enabling the Run state)
- Space (to jump)
- L (to respawn)

The movement keys also adjust the body's acceleration based on whether the "Shift" key is pressed or not. This allows for differentiation between walking and running, which is also reflected in jumping.

3.1.3 Cannon

In this project, the incorporation of CANNON is instrumental in bolstering the game's physics and user experience. To optimize performance, a fundamental strategy has been employed: assigning a mass value of 0 to all objects within the game world, except for the character.

This configuration designates these objects as static, allowing them to remain immobile, while the character, as previously discussed, enjoys the freedom to move dynamically. The CANNON framework has been adeptly harnessed to create associated objects for the obstacles that the character can jump on. These CANNON objects facilitate realistic physics simulations, ensuring that the character's interactions with the obstacles are both convincing and engaging, heightening the overall immersion of players within the game environment.

Furthermore, the character's own physics are enhanced through the integration of a CANNON sphere object, augmenting the realism of its movements and interactions. Through the strategic implementation of CANNON, the project achieves a harmonious blend of dynamic character mobility and static obstacle elements, resulting in a compelling and interactive gameplay experience.

3.2 Camera

The camera in this setup is a critical element for delivering a captivating third-person gaming experience. Positioned strategically behind the character, it adeptly tracks the character's movements, ensuring players have an immersive and unobstructed view of the game world. This dynamic camera system greatly enhances gameplay by providing players with excellent situational awareness and effortless navigation.

It maintains a consistent distance and orientation relative to the character, eliminating the need for players to manually adjust their viewpoint. In summary, the camera's role here is instrumental

in creating an engaging and enjoyable gaming experience from a third-person perspective. We can clearly see the distance and angle it has from the character in the figure below.

Figure 8: Camera Position w.r.t Character

3.3 Lights

Concerning the lights and the overall illumination, we used two THREE.js constructors in order to generate, first, an *Ambient Light* for the entire environment, to which we've assigned an off-white color and the right amount of intensity, and then, a *Spot Light*, approximately positioned in the middle of the map and very elevated from the ground (with a high Y-axis value, so as to reproduce the exposure to sun), which actually helped us to obtain a realistic effect of light reflections and shadows on the model's surfaces.

The Spotlight has the same color of the Ambient Light, but it is significantly more intense than the first one.

3.4 Textures

Other than the ones already included within the imported models, we used and manually inserted into our project two additional textures:

1. A Skybox texture mapped on a cube: to reproduce the environmental sky, we mapped onto a *CubeTextureLoader* object (from THREE.js) 6 images, which (as a whole) pictured a cloudy sky, in such a way that each of them covered one face of the cube containing all the map.

   The complete (un-mapped) texture of the sky can be viewed in the figure below.

2. Character's textures: both usable characters of our game have their own 2D textures, which had to be loaded into our project and correctly mapped on their models.

Figure 9: Skybox Texture (Undivided)

3.5 Menu, Timer and Pop-ups

In conclusion, we added three interactive elements to corroborate our game and to improve its usability.

We built an initial Menu (index.html) in which the player can modify the level of difficulty (enable/disable checkpoints), choose the character he's going to play with and read the instructions about the game's goal and commands. Then, we inserted an in-game Timer at the top right of the player's GUI: it starts when the first movement is performed and it stops when the ending platform is reached, so that each user is able to check how many seconds it took in his run to finish the game. Finally, we inserted specific Pop-Ups whenever a player gets to a checkpoint, in order to let him understand that, if he fails, he can restart from that point using the assigned key, and when he finishes the game, to congratulate him on his effort to have reached the end of 'OnlyUp'.

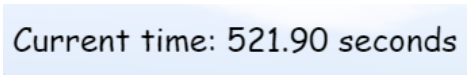

Figure 10: In-game Timer

Figure 11: Checkpoint Pop-Up example