3D Programing

Assignment 3 - Report

Group 4
Catarina Gonçalves 90709
Guilherme Monteiro 90724
Pedro Leitão 90764

For this assignment we had to implement a game of our choice, using Unity. We decided to develop a platform game where you need to go through some obstacles and enemies to complete the game.

We would like to state that everything regarding the platforms and most elements in our scenes were solemnly done by us, except for our main character's model as well as for some of the decorative objects like trees, lamps, bridge and fences, that we downloaded from the Unity Asset Store. Most textures and materials were also downloaded.

1. Our Game and its Mechanics

We wanted to create a platform game that would allow us to play different levels. With that in mind, our inspiration for our project was the well known "Little Big Planet" PlayStation game. Therefore, our main character is **Sackboy** and, in order to complete the game, the player must follow a path and successfully surpass all the obstacles: fire, moving platforms, spikes and enemies.

The game starts with a Start Scene that gives the player the options to play or quit the game.

Figure 1 - Start Scene

If the player wishes to play and presses the "Play" button, the main scene is displayed and the background music starts playing. Our game only contains a level to complete: it is basically an outdoors scenery with several items and street lamps to highlight the path that should be followed. We also added several **checkpoints** (small shepres) throughout our scene to save the player's position so that, in the possibility of our characters dying, the game could simply restart on that previously saved place.

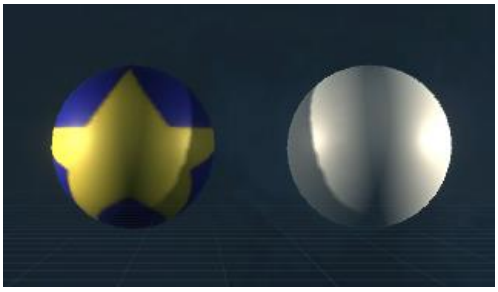

Figure 2 - Checkpoints: Marked vs Unmarked

Initially, we wanted to create at least two different levels but, throughout the development of the initial level, we decided to focus on it and improve all the game mechanics, landscape and details of our scene. In this main scene, we also have the Head-Up Display, where the current score is displayed at the bottom left and the lives counter is displayed at the bottom right. In addition, a mini-map is displayed at the top-left of the screen.

Figure 3 - Head-Up Display

In addition, if the player presses the "Return" Key, a **Pause Menu** is displayed. To create this menu, we added a canvas to our project in order to have a somewhat blurred background and the actual menu with the buttons in the middle of the screen.

As you can see in Fig. 4, the player has several options to choose but the most relevant one might be the inventory since the others are pretty straight-forward.

Figure 4 - Pause Menu

In the **Inventory Menu**, all the outfits picked up by the player throughout the game-play are displayed and it is possible to choose one of them to wear. It is also possible to delete the items from the inventory by pressing the "X" button at the top-right of each inventory-slot.

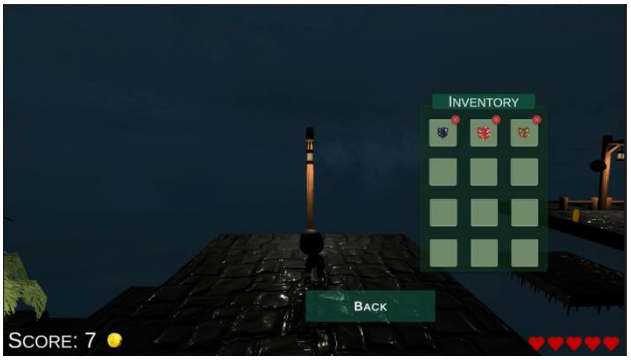

Figure 5 - Inventory Menu

Then, there are two possible outcomes: the character loses all 5 lives and it's Game Over (Fig. 5) or the player finishes the level (Fig. 6). In both these End Scenes, we display the total acquired score and some buttons depending on the outcome.

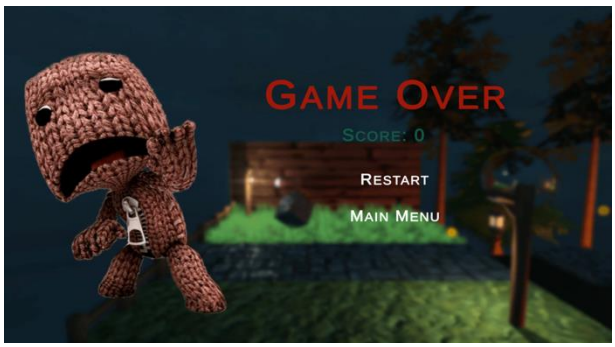

Figure 6 - Game Over Scene

Figure 7 - Completed Level Scene

2. Sackboy

To have Sackboy as our game character, we got the model from the original game. He is able to walk, run, jump and pick coins and different outfits. We added several animations to it like constantly looking around when he stops walking, when he dies and some more, using the Mixamo Adobe Platform.

Each time he picks up an object, passes through a checkpoint, chooses an outfit from the inventory or when he dies, you can hear a sound that matches the action made. For instance, when he catches a coin, a coin-like sound can be heard.

It is relevant to state that our character only has 5 lives: if he dies either from getting shot by an enemy, falling from a platform, falling on spikes or being near an explosion, he loses 1 of his lives and the game restarts at the previous marked checkpoint. Once the character loses all 5 lives, it's game over.

Figure 8 – Sackboy with different outfits.

3. Collisions

Our character is able to push objects or pass through them in order to collect them. In both cases, we added a collider component to the object in question but only in some did we set the isTrigger argument. For instance, when it comes to collecting the coins, we defined an OnTriggerEvent function that sums 1 to our score and destroys the object at the end (since once the coin is picked up, it does not make sense to keep it in the scene). The same applies to our Sun/Moon objects but, in this case, we change the skyBox from day to night or vice-versa, depending on the collected object, and to our Outfit Bubble objects (1)*, where we destroy the object in question but store it in our inventory. We did include other OnTriggerEvents like the explosion buttons that release a cylinder explosive that the player must kick in order to knock down a wooden wall.

1. These objects consist of a Glass Sphere with a Cube with the respective outfit texture inside it.

When it comes to autonomous objects, we added some moving: some move on the x-axis, others on the y-axis and others on the z-axis. (Fig. 9)

Regarding static enemies, we couldn't find models to our liking. We ended up creating a cylinder and gave it a simple texture (Fig. 10). Once our character reaches their range, these enemies are constantly shooting projectiles every frame (small cubes with the bloom effect) in the direction of Sackboy's position. If a projectile hits our character, he dies and loses one life. It is worth mentioning that each projectile disappears 3 frames after being shot.

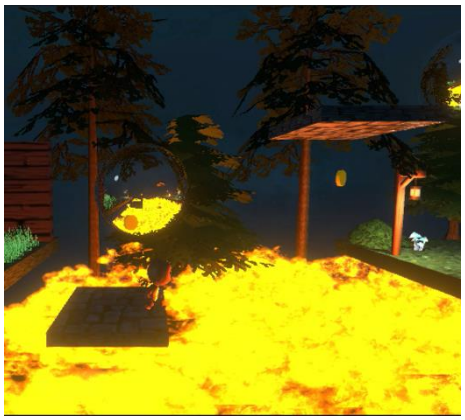

Figure 9 - Moving platforms

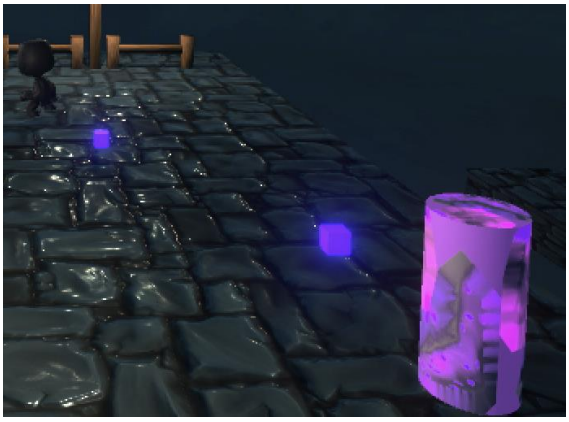

Figure 10 – Enemies shooting projectiles

In addition, we also added some fire cylinders that fall from the sky and end up rolling in an inclined plane that our character is trying to climb. This was done to increase our game difficulty since the cylinders kill our character when they hit him.

4. Visual Effects

In our project, we used the Built-In Renderer Pipeline. However, this ended up being a challenge that we did not foresee. We did try to change our project to the Universal High Definition Renderer Pipelines but most of the textures and materials in our scene were not compatible with any of these pipelines.

4.1. Bump Mapping

While decorating our scene, we found some stone and wood textures from the Unity Asset Store. We added the stone texture to our platforms and the wood texture to some planes and cubes in our scene. To enhance the bumps, we changed the values of the material's Smoothness, Height Map and Occlusion.

Figure 11 - Bump Mapping

4.2. Environment Mapping

Regarding reflection, we created a sphere and added a reflection probe, positioning it inside the sphere. To get the desired reflection, we had to create a new material with the Metallic and Smoothness properties set to max (1).

On the other hand, when it comes to refraction, we know it would have been easier to implement it using one of the other pipelines available but, since the updated version of the Shader Graph only renders materials on the Universal or High Definition Render Pipelines, we could not create

the refraction from scratch but we did try. However, after much research, we discovered that the Standard Assets of Unity included a Refraction Shader as well as a colorful glass texture and that was what we used in our project.

Figure 12 - Reflection

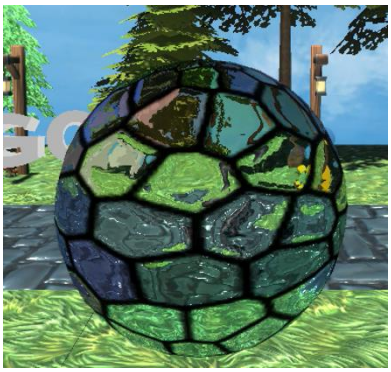

Figure 13 - Refraction

4.3. Lens Flare

Since it was not possible for our player to move the camera, we decided to add lens flare to our Sun and Moon objects in order to make it as realistic as possible. To do so, we added a point light and set the flare argument with some of the lens flares we got from the Standard Assets of Unity.

Figure 14 – Different lens flare used

4.4. Billboards

As billboards we added some grass at the beginning of our scene and some text throughout it. To do so, we wrote a simple script that changes the position of our sprites according to the camera's position. In order to see how this works, we advise you to look at the mini-map.

Figure 15 – Billboards

4.5. Post-Processing

Regarding post-processing, we implemented the Bloom Effect. To do so, we added a Post-Processing layer to our camera and created an empty object, *PostProcessingManager*, to which we added a Post-Processing volume and a Bloom component. To both of these components, we changed their layer to the "Glow" layer that we created. Then, we created new materials where we set its emission according to the color and intensity desired and added them to a few cubes that we positioned throughout the scene.

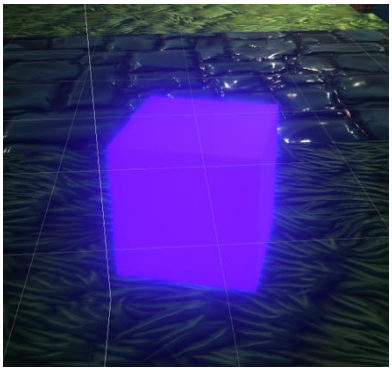

Figure 16 - Bloom effect

4.6. Particle Systems

In order to have particle systems, we used a free Asset, "Unity Particle Pack"- Unity Technologies, from Unit Asset Store to simulate fire and explosions. These effects were used on the Fire Platform, we changed the "FireBall" prefab in order to get what we wanted and used it 25 times to simulate a real blaze. We also used the "SmallExplosion" prefab on the cylinder that simulated TNT and the "FlameStream" prefab on the on-fire cylinder. All these prefabs were deployed using the "Instantiate" function.

Figure 17 – Particle Systems

4.7. Global Illumination Effects

To our PostProcessingManager GameObject, we added an Ambient Occlusion component. In addition, we changed our lightmapping Settings and set Ambient Occlusion to true.

In regards to lightmaps, we set some of our objects and prefabs to contribute to global illumination and receive it from ligthmaps.

When it comes to real-time/baked Lighting, we had our street lights set to different light modes: some baked and others real-time. We could have made one of our reflective spheres baked and that would be noticeable while playing the game, since the sphere would only reflect the static objects of the scene. However, the lights can also be tested since some will light up our character while the baked ones won't.

5. Extra functionalities

When it comes to extra functionalities, we believe that we were very creative and increased our game's complexity with the checkpoints, spikes and moving and disappearing platforms. In addition, the rolling on-fire cylinders were also a great asset in our game, intensifying the level's difficulty.

Regarding the appearing/disappearing platforms, we created a simple script that displays the next platform.