Interactive Graphics Project

The aim of this project is to create a small tech demo inspired by the style and themes of the *The Legend of Zelda* games. To achieve this, a very basic combat system was designed and the models were made out of geometric shapes in order to preserve the low poly style of the initial entries of the famous franchise. Finally, two different levels were created with the aim to replicate the feeling of exploration and adventure that are so characteristic of the games.

In the following sections I will detail the different aspects and functionalities of the project.

Description of the environment

As the scope is reasonably ample and potentially complex to execute, I decided to use three.js to mitigate efforts directed to basic functionality such as defining geometric shapes vertex by vertex, implementing lighting models and others. The file will be imported by the index HTML code as a module, and this combined with the fact that some additional images will need to be loaded mean that it is necessary to set up a local server in order to run the game.

The lighting model used for most of the models will be the Phong model through the Phong material offered by three.js, while the elements of interface (the life gauge), the flame of the torches and the cave entrance will be basic materials that simply display a colour to make them as intense and clearly visible as possible.

Additionally, the lighting setup includes an ambient light that radiates some light grey light and a directional light coming directly down that has a higher intensity and emulates the sun. The torches also emit light when they are lit.

As for the folder structure of the project, in the root folder the index HTML file, the main JS file, README file and the folders for images and libraries are found. As for the libraries folder, it contains the *Three.js* and the *Tween.js* library code, while the images folder contains textures that will be discussed in the following section.

Libraries and external assets used

- *Three.js*: used for all the rendering purposes, geometry definition, texture mapping be it colour textures, normals or bump maps, material definition, light model definition and graph scene presentation, which includes the ability to define hierarchical models.

- *Tween.js*: staple of all animations present in the project, the feature of tween chaining also allowed for definition of cyclic animations.
- Other assets: contained in the images folder, they are a series of images taken from the web for the purpose of texturing different objects and or adding complexity to them.

  The list of normal textures is the following: *hairNormal.jpg*, which is used in the character's hair, *stripes.png*, for the torches, and *tiling.jpg*, for the dungeon's floor.

  Then we have the colour textures, being *arches.png*, for the dungeon's walls, *enemy.jpg*, for the skin of the enemies, *face.png*, applied on the front of the hero's head, *grass.avif*, that covers the overworld's surface, and *wood.webp*, used in all kinds of wooden surfaces.

  Finally there is a single bump texture, *fabric.avif*, which is used to give the character's robes a fabric like surface. Also important to note is the fact that, since *Three.js* allows the use of colour textures for bump map assignment, in the case of the grass material and the dungeon walls material the colour texture was also assigned as the bump map texture in order to create a sense of depth in an otherwise flat texture. This works because the shadowed areas in the texture are interpreted as lower height areas whereas vibrantly coloured zones are interpreted as higher, which is an acceptable behaviour in both the grass and the arches textures that produces a convincing effect.

Technical aspects

In this section we will consider all different aspects of modelling and animation, as well as how the game loop was implemented, the progression and other systems in place.

- *Game loop and progression*: there are two main scenes or levels in the game. The first one, that gets loaded every time you start, is a grass plains land with some mountains in the horizon and an entrance to cave surrounded by trees. At this point the player is expected to receive the sword offered by the elder before heading into the cave entrance. If he does otherwise, then he will perish because without the sword the enemies inside of the dungeon will not take damage upon his attacks.

  After heading into the cave entrance, he will be positioned inside the view of the camera and the layout of the scene will change a lot. The dungeon has four torches placed near the top centre, four walls and the floor. From the floor various enemies will start to emerge in short sequence and will start to try to kill the player, which will restart the game to its initial state. If, however, the player starts to defeat them, the torches will begin to light up according to the proportion of enemies dealt with and after none remain a Triforce prop will come down from the ceiling. After receiving it the game will again restart as it has been completed. The restart was simply implemented as a page refresh.

- *Combat system*: when the user performs an attack action, an event listener checks if there is any enemy in the appropriate distance with respect to the player and if the orientation of the player with respect to the direction where the enemy is coming from have an angle of less than 45º. If these conditions meet, then the health points of the enemy are lowered by one point and their *getting hit animation* plays, propelling it backwards from the hero.

  Likewise, when the enemy gets too close to the hero, the hero will play his own *getting hit animation* while being propelled away from the enemy. In the case he lands on another enemy, it can instantly spell doom for the hero since the enemies will play a sort of ping-pong with him until he runs out of health points, which lower one point each time he gets hit.

  Both enemies and the hero have three discrete health points, this is okay because defeating an enemy is very easy as they move noticeably slower than the player.

- *Modelling and hierarchical structures*: let us start with the most complex object of the game, the hero, which will be controlled by the player.

  The body is divided into the abdomen and the chest, the first of which is the root of the model and which is a modified box geometry that has a wider bottom half so as to depict the more loose lower half of the hero’s tunic. Then, the upper legs and upper arms get assigned as children of the abdomen and chest respectively and also become parents of their lower counterparts of similar shape. These parts belonging to the extremities are shifted in the vertex level so that when they rotate, they do not do it around their geometrical centre but around the joint, and the legs will feature boots at their ends. For the head, two icosahedrons of similar size were used, one for the actual head and the other for representing the hair which is one of the two children of the head. The other one would be the hat, which is composed of a translated cone that instead of a flat base has a sphere as its parent. It was translated in order to make it rotate correctly around its base for the purpose of animating the walk cycle.

  Then we have the elder, which borrows much of the structure of the player with the following modifications: the abdomen was enlarged until it touches the boots, the hair was moved to make it seem like he has a receding hairline, he has no hat or legs and instead of upper and lower arms he has arms and hands. He was also made to look weaker than the player by tilting his chest forwards slightly and making his chest body part smaller.

As for the enemy, its design was inspired by the *Chuchu's* design from the *The Legend of Zelda* games. The idea is that it is an elastic blob that has a base, modelled by a small, squashed icosahedron, and a head, modelled by a bigger icosahedron, both connected by a cone geometry which base connects with the head having the same radius.

Now there are four decorative objects to go over, such as the trees, which consist of a cone making the trunk of the tree which goes through two icosahedrons symbolizing the leaves. The torches, which are cylinders generated with such a low quality that they become prisms smaller at the top than the base, have a flame that is represented by a bright red small cone on its top and the sword is made of a cross guard and a blade, both made from two components. There is also the Triforce prop, which is composed of three three sided cylinders disposed in a triangular pattern.

All the models relating to the dungeon, walls and ground, are simple quads and the entrance to the dungeon itself is a stretched circle geometry. Another example of circle geometry is the life indicator in the top, but the number of vertices in them was reduced to five for congruence with the rest of the low poly style.

Finally, the terrain is a bit more complex in its geometry than the rest of the objects. I would like to call attention to the fact that there is a radius with relation to the centre of coordinates in which the height is zero, and after that the height function applied to it is a function where sine and cosine functions multiply each other creating small hills separated by a regular interval in every direction. This was done by applying a function to each vertex where their x and z coordinates where left as they were, and the y coordinate was changed.

- *Animation system:* firstly, a way to define keyframes and the transition between them ought to be defined. Tween comes into play in this department and the way in which I solved this was to first define a group of tweens that each acted on a body part and had in their names the number of said keyframe. Then, for each tween associated to keyframe *i*, it would be chained to a tween belonging to the following keyframe *i+1*, in such a way that when starting at the same time the keyframes for *i*, then all the following keyframes would follow.

  Taking into account the fact that all tweens inside a keyframe need to be assigned the same interval time for this to work and that you can define animation cycles by chaining cyclically the tweens from different keyframes together, this is the approach that I took in order to the define all animations of the following in game objects:

The hero has three animations. The first two ones are very simplistic, like the breathing animation, which is just a cyclic scale transformation that makes the character look like he is breathing. The second one is the attack animation which depicts a quick, aggressive and very rigid motion. Lastly, the walking animation is more interesting to discuss, as it is the most complicated animation and exploits the complexity of the hero's model even though it is entirely rotation based. There are five keyframes, and even though the first and last are the same, this is needed for chaining the fifth keyframe to the second keyframe and dedicating the first one to the transition between rest and walking. In a cycle, the upper arms will rotate in opposite directions to each other, while the upper and lower legs also do the same motions but in such a way that one foot is placed in front of the other, bending the knee to better simulate a human like stride. Finally, the comically long hat that trails the hero's movement dangles with his motion rhythmically. He has also an animation for getting hit, where he gets propelled backwards and his tunic become reddish for a moment.

Moving on to the elder, he starts in a static position offering the sword to the player. When he hands it over, an animation will play that will take him from the centre of the stage towards the back of it, removing him from the reach of the camera. Additionally, his shoes will alternate in movement cyclically to make it seem like he is moving himself and he will position his hands, by rotation of the arms, as close to the centre of his back as possible without clipping his geometry inside his body too much.

The enemy is another notable aspect of the game. Its animation cycle tries to be as chaotic as possible, by portraying its movement as a byproduct of the contortions of its own elastic body. For this, rotation in both the x and z axis as well as scale transformations on all axis that simulate how it squishes were implemented through a custom function. As the enemy never stops to chase the player, until it dies and disappears, I defined this animation as a cyclic animation that starts at the beginning of the game and never stops. Secondly, it has an animation for when it gets hit, which makes it become reddish and then normal again at the same time that it gets propelled backwards from the player at high speeds.

Finally, there are three animated decorative objects, the torches, the trees and the Triforce. The trees will rock cyclically to the wind in such a way that only the leaves rotate in the z axis and get rescaled in the x axis, their movement is always apparent as they are always facing the camera which is looking in the -z direction. The torches, on the other hand, start being static objects and once enough enemies are eliminated they light up with an animated flame that is simply cyclically scaled in the y axis. Finally, the Triforce will come down from the sky in the dungeon while it rotates when all enemies are removed.

Implemented interactions

Let us speak of the concept of interaction as the direct recognition of inputs from the user, in which case three different types of interactions can be recognized.

- *Difficulty level setting:* a pulldown selection menu that appears on the top of the page allows for setting the difficulty of the dungeon stage, in terms of enemy numbers. Easy corresponds to four, meaning each time you defeat an enemy a torch will light up, while medium means you will face eight enemies such that two enemies correspond to a torch lighting up. For the hard mode, you will be facing sixteen enemies which will amount to four per torch.

  In order to apply your difficulty level selection, you can touch the button on the right of the menu and the page will be refreshed with a new parameter expressing the difficulty in the search query, to be processed by the JS file.

- *Click event handling:* an event listener was set up, linked to the *click* event. When such an input is registered the logic carried out is the one explained in the combat system part of the technical aspects section, when the hero performs his attack sequence.

- *Movement keys event handing*: two event listeners were set up for the *keydown* and *keyup* events respectively. They both act on the W, A, S and D keys of the keyboard and have the objective of allowing the player to move. The first of them both sets the direction of the movement in each component to one, triggers the walking animation and makes the body of the hero look in the direction of movement. The second, on the other hand, sets each component of the direction to zero and stops the walking animation, restoring the resting position.