**You shall not run!**

Paolo Roberto Accaputo 1916925

Giancarlo Campeti 1905580

26th September 2021

**INDEX**

**1 – Libraries and technologies**

**2 – About Gameplay**

**3 – Gandalf Hierarchy**

**4 – Animations**

**5 – Textures**

**6 – Lighting**

**7 – Audio**

**8 – Collider system**

**9 – Basic Logic**

**10 – User Interface**

**1 – Libraries and technologies**

To start things off we are going to discuss the technologies used to develop *“You shall not run!”.*

We agreed on using THREE.js, a high level javaScript library mainly for two reasons, it is very simple to learn and use, and it takes advantage of the pipeline architecture of the GPU.

It is based on WebGL and has a very high compatibility making so that almost every browser can run it.

THREE.js has three main components, the scene that contains every other object, the camera which handles the point of view of the user and the renderer which draws the scene into an HTML Canvas.

Then we had to decide how to handle collision detection and we found THREEx.Colliders, another easy and user-friendly library.

The last decision to be made was about the models to be imported into the project, after a brief research online we found out that GLTF models, based on JSON, are optimal for web implementations so we used the GLTFLoader API to import such models into the game and setting up meshes, textures, hierarchical structures and so on.

**2 – About Gameplay**

Our game is clearly inspired by “The lord of the rings” with a retro arcade touch.

It is basically a side scrolling endless runner, in which you control Gandalf, and your goal is pretty simple, you have to survive avoiding various threats that will try to harm you and at the same time trying to collect as many coins as possible to raise your score and maybe set a new record!

Gandalf will run by default, and he can perform two different actions in order to avoid two kinds of threats:

* **Jump –** to avoid the hot cauldrons that populates the swamps of Mordor
* **Fade –** the best way for a Wizard to avoid the swarm of arrows that the orcs will shoot at him is to just fade away into another existential plane!

While running throught the swamps trying to stay alive while collecting as much gold as possible, Gandalf should keep his eyes open for Stars, these special collectibles will be very helpful in harsh situations granting a complete invulnerability for a brief period of time.

One does not simply walk into Mordor, it is a dangerous place for sure!

But for the bravest players who find it not so much of a challenge, we implemented two additional level of difficulty, be careful because the run will be a lot faster and even more threats will spawn to hunt you down!

**3 – Gandalf Hierarchy**

The model we choose for Gandalf has a pretty simple hierarchical structure: the root node has 3 children, the torso and the two upper legs, the torso is father of the head and the two arms (divided in upper, forearm and hands), while the lower legs are obviously children of the upper ones.

This hierarchy is pretty simple but at the same time it was all that we needed to implement the animations that we wanted, to handle them we used the .traverse() function that goes through the whole hierarchy and makes it so that transformations applied to a particular bone will be inherited by all of his children.

**4 - Animations**

Every object in the scene share a basic animation: the translation along the x Axis in order to simulate the endless run of our favourite Wizard.

Gandalf is the only model subject to a more complex set of animations for obvious reasons.

* **Idle –** the name may be tricky but this animation consists in a pretty straight forward run, almost every bone of the model is involved in this animation and it is the “root” of the animation graph, meaning that every transition will start/return from/to idle.
* **Jump/land –** this animation was more complex to implement that we originally thought, it consists in three phases: jumpFix() in which we set the correct leg position to start the jump phase, Jump() which is self-explanatory, Gandalf jumps in a retro fashion with arms and legs wide open, and to round things up we got Land() which will bring Gandalf back to the ground and to the idle animation.
* **Fade/Unfade –** this animation modifies both the bones and the material of the model, Gandalf will become transparent in what looks like a spiritual form while raising up his arm and enter a levitation state for a few seconds after which he will resume is physical state and get back to running.
* **Blink –** to bring back the classical “oh I took damage” feel of retro games we made so that Gandalf will briefly blink every time something hits him.

**5 – Textures**

In order to handle texture management we used THREE.TextureLoader,

An easy tool that THREE.js provides to us all we had to do was to apply an RGB encoding to make the colors appear more vivid and a 16x anisotropic filter to remove the blur that came by default with this loader.

The textures we used in the game are mainly for the HUD and the scrolling background, the only “Complex” texture to manage was the one related to the invulnerability state which turns Gandalf into a golden wizard and back to normal after some time.

**6 – Lights**

We used two light sources in our scene, THREE.DirectionalLight() to simulate the sun vertical rays coming from an infinitely far away point in the space and THREE.HemisphereLight() which coupled with the directional one provides a more realistic feel to the environment with a decrease in intensity from sky to ground.

**7 – Audio**

To handle audio in THREE.js we had to learn to use three main components:

THREE.AudioListener() which attached to the camera handles the positional audio, THREE.AudioLoader() which is used to load sounds and binds them to variables in order to be used and finally some mp3 samples to be imported and manipulated.

**8 – Collider System**

To handle collision detection and interactions we used THREEx.ColliderSystem , an helpful library that made our work smooth and easy, all we had to do was to give to each model in the scene that needed to interact with Gandalf a proper collider (a box of set dimensions to resemble the specific model) and an helper to make these boxes visible to help during tests.

**9 – Basic Logic**

Both the speed and the number of threats are functions of the difficulty level that the player can set before each run.

During a run different objects will spawn randomly, some are useful, some harmful and others are there just to provide a more complex background with some 3D models rather than a plain texture:

* **Threats –** the two different threats (cauldrons and arrows) are stored into an array, during the game the function spawnThreats() will randomly choose one of the two and position it into the scene, with the correct collider, helper and specific sound.
* **Trees –** these are 3D models that serves more as a background and that’s why they don’t have a collider associated, Gandalf cannot interact with them in any way, we just position the model in the scene via the spawnTrees() function.
* **Coins -** coins spawn at a fixed position along the y axis and are randomly generated in groups of one, two or three. They obviously come with a collider associated in order to make them interactable, Gandalf needs to jump to reach them and once he enters in contact with a coin the latter will disappear from the scene and the score will be updated.
* **Stars –** same concept as coins but these special collectibles will spawn more rarely and at a higher y position, requiring a more precise timing for the jump. The stars comes with both a collider and a texture, the collider is there to make it so that Gandalf can pick them up while the texture is the same that will be applied to Gandalf during the invulnerability time.

The whole logic is controlled by the update() function, similarly to the update routing used in C# this function will be executed continuously during the game and will take care of positioning objects and colliders, handling the transitions to the various animations of Gandalf and controlling the game state, in particular when the player ends their lives the gameOver screen will be launched.

**10 – User Interface**

Every part of the game is associated to a DOM element in the HTML file, some are inside the index.html file and some are created via javaScript functions at runtime like homeScreen() or gameOverScreen().

During each run there is a UI displayed which informs the player about the remaining health and his score both updated in real time.

When the game is launched the player will land to the home page in which he can access a simple tutorial slideshow as well as the difficulty settings and a button to start a run.

When the player loses all his health points the game will end and the game over screen will pop up, in this screen the player can see his final score, start a new run or return to the home page.