Final Project - Zombie Runner

Interactive Graphics

Donato Emanuele Mele 1753058

June 7, 2022

1 Intro - How to play

Zombie Runner is an infinite runner. The player should not hit zombies and obstacles. To avoid zombies the player has to move to the right or to the left. To avoid obstacles the player has to jump. The player has 5 lives and he will loose one of them each time an obstacle or a zombie is hit. The score is based on the time passed until the player expires all his lives. This game took inspiration from Subway Surfers, a mobile game in which the aim is the same but there are trains instead of zombies.

Figure 1: Game menu

2 Tips

In the main menu it is possible to select different options with respect to the user preferences. In fact the user can choose among two types of player, two possible scenarios and also some graphics options. Using the graphics options the user can set the right combination between quality and playability. It is suggested to always play at 60 FPS (there is a counter in the playmode). It is required to disable antialias and to lower the resolution in the Options panel if the game is not getting 60 FPS. This may be required if the scenario is set to Night. In this case a lot of pole lights are enabled resulting in an FPS drop. Once the game starts at the left down corner it is visible

a FPS counter. There is also a loading indicator and a timer of 3 seconds after the loading. This timer is usefull to let the user get ready before playing.

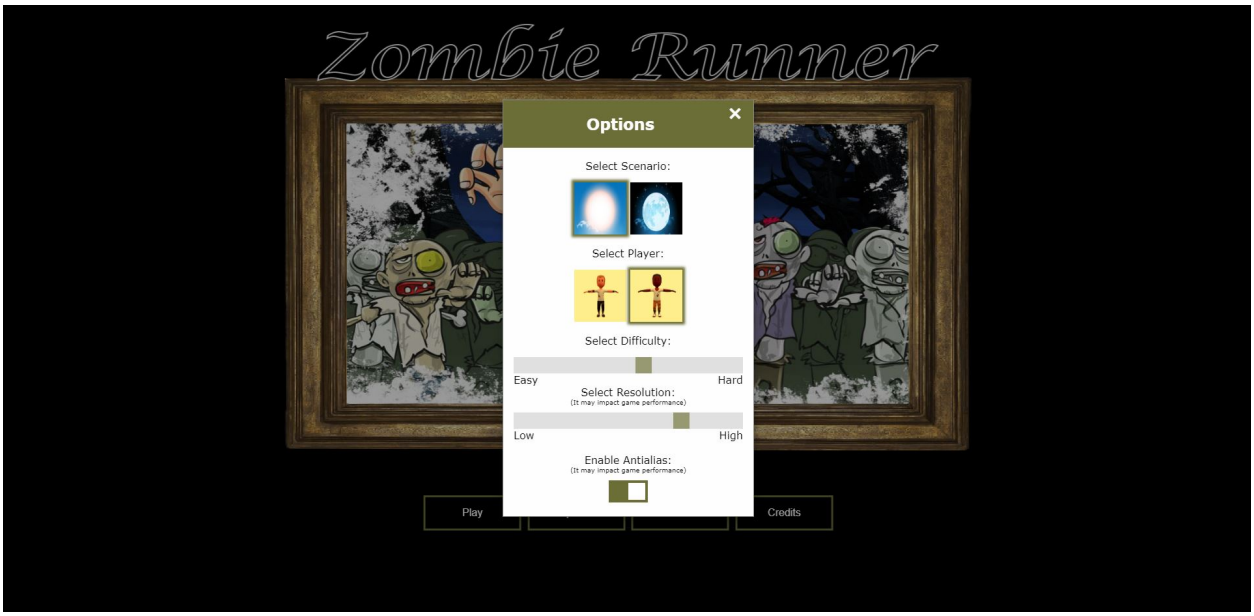


Figure 2: Game options

3 Commands

- **Move**: Use the arrow keys.
- **Jump**: Use the space key.

4 Libraries and tools

To develop this game I used **Three.js** which is a JavaScript library that uses WebGL. Three.js allows the creation of GPU-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is done thanks to WebGL, a low-level graphics API created specifically for the web.
For the animations **Tween.js** was used. Tween.js is a simple but powerful JavaScript library for tweening and animating HTML5 and JavaScript properties. Without this library it would have been much more complex to make animations. It is possible to choose different easing function to perform smooth animations. A lot of different properties can be animated by using

the chaining commands.
To load the 3D models the **GLTFLoader** was used in addition to **DRACOLoader** for more complex models. There are about 20 models loaded; for this reason the Loading Manager of Three.js was used to keep track of the loading and to display a percentage to the user. GLTFLoader was a good choice since the chosen models are in Gltf and Glb formats. This library has callbacks to check when an import is over. DRACO loader was used for the zombie and player models. These models were compressed using DRACO, so in order to decompress them the same library was needed.
To display the FPS counter the **stats.js** library was used. This library was fundamental to optimize the game. Along with the **OrbitControls.js** library that lets add really simply the camera movement which is really useful especially to debug colliders. Regarding the colliders the **Threex.js** library was used. Using this library helped a lot to implement and test the colliders. The logic behind colliders was to use a box to simulate the collider of a more complex object. The zombie, the player and the obstacles followed the same principle; they all have a box collider. The player checks if some colliders enters in his box collider and if this happens a player life is gone. To test this behavior, box helpers were used thanks again to the threex.js library.

Figure 3: Day mode

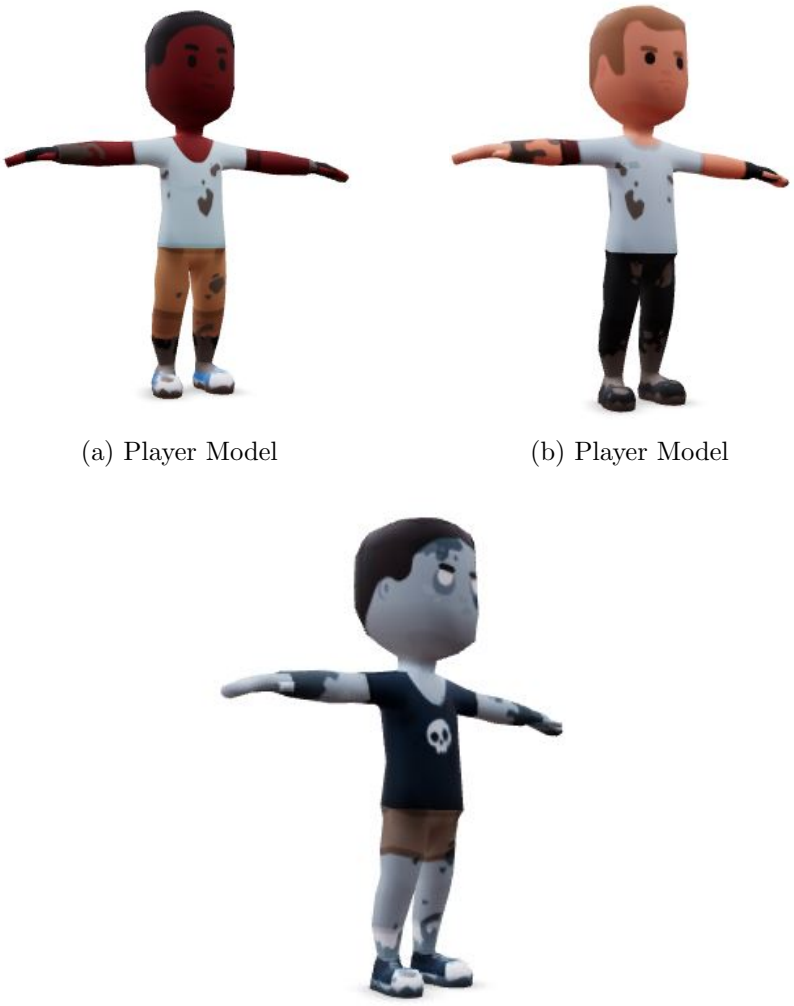
5 Models, lights and textures

A special thanks goes to the **Kenney.nl** website that provided all the 3d models for this game. In particular trees, houses, players, zombies, obstacles, poles and the road. All the models have been downloaded in .glb format which is a binary format derived from .gltf. Just the player and the zombie models have been downloaded directly in the gltf format, this was needed for the animations. In fact the animations are done using the nodes of the gltf file which is a file that contains different information on the models as the nodes that made up the object, the materials etc.. This file is in JSON format, making it extremely easy to link the nodes in the file with the .js game. The player and the zombie model are complex objects. They have a lot of parts from the toes, the knee to the shoulder, the neck and the head. Not every node was used for the animations but just most of them including the legs, arms, forearms, upper legs, chest, the neck and the head.
The downloaded models have different textures attached to them. The only model not downloaded was the floor made using Three.js. To make the grass, a texture normal map was used together with the green color.
To build the sky the technique of the skybox was used. The skybox is a cube mapped from the inside thanks to 6 different textures. This perfectly accomplish the sky result.
To light the scene a directional light was used. This light changes its properties based on Night or Day. In the night mode it is darker and emits a skyblue color that reminds the moon reflection. In the day mode the light is brighter and emits white light. Also in the night mode it is possible to see thanks to pole lights. In fact there are ten light points each of them emitting a pole light that moves together with the street. The point lights have an high decay to better simulates pole lights.

Figure 4: Night mode

6 Animations

As said the Tween.js was used to perform the majority of the animations. The only animation not performed in this way is the floor movement. Despite what the user can feel, the player and the road are not moving and the environment is always moving. There are two floors one follows another, when the first reaches the player it is placed right behind the other floor in an infinite loop. The models like the houses, the pole lights and the trees are randomly placed on the floors and then they follow the floor movements; in fact they are floor children. The same does not hold for zombies and obstacles because they change their position randomly each time and not each play as the others models.
The complex animations built in chaining functions using Tween.js are for examples the player jump in which the player rotates his chest and then raises his legs. Obviously the animation is really complex. In fact legs, arms, forearms, upper legs, chest, neck and head are rotated or translated. The same holds for the running animation where the player left arm is synchronized with the right arm to make the feeling of a walk. A different animation, but with the same principles, is the zombie one. In this case arms are in a zombie pose and the zombie is in an idle pose in which his legs appears as nearly broken. An easier animation to implement was the one when a player



(a) Player Model

(b) Player Model

(c) Zombie Model

Figure 5: Complex hierarchical models

hit something, in this case all the player nodes change materials opacity to a certain value giving the effect of a sort of flashing.

7 Interactions

There are different interactions in this game. In the menu Options panel it is possible to select the player, the scenario, the game difficulty, the resolution and if enable or not the antialias. Antialias is a technique for minimizing the distortion artifacts (aliasing) when representing a high-resolution image at a lower resolution. Antialias and resolution may impact the playability as

they can decrease FPS. All this interactions are saved in the localStorage and then are used in the javascript file to update the game variables giving the desired effect. Changing the difficulty changes the speed in which the floor is moving and also the width of the obstacles. Night mode enables pole lights and changes the color and the intensity of the directional light. Day mode dis-activates pole lights and changes the color and the intensity of the directional light. Changing the resolution means to decrease or increase the aspect ratio factor that will give more or less pixels. The more pixels are visible, higher is the resolution. An high resolution may impact game performances. To make an example with maximum resolution (so with aspect ratio of 1), the game in night mode is counting more or less 17 FPS in a computer with no dedicated graphic card. This is not a good result of course but just by reducing the resolution and disabling the antialias a better playability can be obtained. In day mode the result is pretty stable since there not pole lights which are really expensive. When the player dies a menu appears letting the user to try again the game or just go to the main menu. Others interactions not related with the game variables are the credits and the commands section.

Figure 6: Game over