# *ZOMBIE SURVIVAL*

## *Interactive Graphics Project*

***Nicolò Mantovani,1650269***

***Gabriele Nicosanti,1599265***

## 1. Introduction

The aim of the project was to create a 3D game through a library named '*three.js*', one of the most popular JavaScript framework for displaying 3D content on the web.

We developed a first person shooter game whose rules are simple: the player is surrounded by a fixed number of zombies and, in order to save its own life and win the game, he must kill them all before time runs out. If the player is touched by a zombie at least one time then the game is ended, otherwise the player can freely run within the map bounds aiming to shoot the zombies. The zombies constantly look for the player so it is not possible to escape them for too long.

## 2. Levels

We have arranged four levels of difficulty (easy, medium, hard and madness), each in a separate JavaScript file, to let the player have a wider game experience and a brief tutorial that shows how to play the game. There is also an extra mode called Dark, as difficult as Easy mode, where the hemisphere light that simulates the sun has been removed and the environment is enlightened only by the lamps that turn on whenever the player gets enough close to them. The ambience really puts the player in the apocalypse mood, but unluckily it is computationally heavier than the other game modes, that's why it has just been added as an extra and all the lamps have a light texture instead of a real light.

Each level has a timer and if the player does not kill all the zombies before the countdown he loses. The difficulty of each level depends on many factors:

- Timer: in easy the timeout is set to 150 seconds while in madness is only 100 seconds, so in harder levels you have less time available;
- Number of zombies: in easy we have 10 normal zombies, 1 Hulk zombie and 1 Giant zombie, while in madness there are 30 normal zombies, 7 Hulk zombies and 3 Giant zombies.
- Zombie life: in easy are required less bullets to knock down zombies, 1 bullet for normal, 2 for Hulk and 5 for Giant, while in madness 5 for normal, 10 for Hulk and 30 for Giant.
- Zombie speed: the harder the level, the faster the zombies.

| | **Timer** | **Normal Zombies** | **Hulk Zombies** | **Giant Zombies** | **Zombie Speed** |
|---|---|---|---|---|---|
| *Easy* | 150 s | 10 (1) | 1 (2) | 1 (5) | 0.03 |
| *Medium* | 150 s | 16 (1) | 3 (5) | 1 (10) | 0.04 |
| *Hard* | 100 s | 30 (2) | 7 (7) | 3 (15) | 0.05 |
| *Madness* | 100 s | 30 (5) | 7 (10) | 3 (30) | 0.06 |

*Levels info table: the values in brackets are the zombies' life points.*

In general Giant zombies move 2 times faster than other zombies. When there is only one zombie left its moving speed doubles as a last chance for zombies to defeat the player.

The game starts in a pause menu and during all the game experience at the top of the page are shown the remaining time, the current score and the number of zombies left. The score is calculated using the life of the zombies, so you get as many points as the life of the zombies that you killed.

Zombies are spawned all at once at the beginning in a random position in the map, but there is no chance that they spawn to close to the player so he cannot get instantly killed without playing.

## 3. Controls

The tutorial page fully explains the controls of the game. It is possible to move inside the game using the WASD or the arrow keys like most of the videogames and control the camera rotation using the mouse. As a design choice, it is not possible to move the camera vertically but only horizontally. Holding the shift key down lets you run. The cursor is only visible in the pause menu and inside the game the cursor is hidden for a better immersion. It is possible to shoot bullets using the space bar or the left mouse button, and holding them lets you shoot in bursts. The right mouse button is used to position the gun in front of the player, simulating the aiming action of bringing the gun in front of one eye.

The controls are managed by the **FirstPersonControl.js** that takes inspiration from the one in the examples controls folder of the GitHub repository of **three.js** (https://github.com/mrdoob/three.js) but it has been highly changed, especially the use of the mouse to move the camera. In fact, the original one used the position of the cursor to turn the camera, with a rotation speed based on the cursor position on the screen, while ours is based on the pointer lock event so the camera moves just as much as the mouse moves, like in many videogames, and without showing the cursor on screen.

## 4. Three.js

Three.js is a JavaScript library and an API (Application Programming Interface) that is used to create and display animated 3D computer graphics in a web browser without relying on proprietary browser plugin: this is possible due to the advent of *WebGL.* Three.js runs in all browsers supported by WebGL 1.0. The version that has been used is the *'r105'* released in May 2019, and some of the main features of this API are:

- Scenes: add and remove objects at run-time;
- Cameras: perspective, orthographic, cube, array and stereo;
- Geometry: planes, cubes, spheres, torus;
- Objects: meshes, particles and more;
- Materials: basic, Lambert, Phong, smooth shading, texture and more;
- Lights: ambient, directional, point, spot and shadows;
- Data loaders: binary, image, JSON and scene.

The three.js GitHub repository is full of ready to use useful examples like the loaders, so we used the **MTLLoader.js** and **OBJLoader.js** to load the gun model.

## 5. Scene, environment and camera

Through the method *'Scene()'* it is possible to create the scene, which allows to set up what and where is to be rendered: objects, lights and cameras.

We have placed a texture, as background, that represents a red/orange sky and a red fog inside the scene in order to recreate the post-apocalyptic atmosphere. The player holds a weapon, whose model has been imported from the 9th demo of the GitHub repository **threejs-demos** (https://github.com/saucecode/threejs-demos). The weapon constantly floats up and down in front of the player to simulate breathing. It is also possible to position the gun at the center of the screen to simulate the aiming at the enemies.

The environment in which he can move is composed of buildings, sidewalks, lamps and the ground. Each of them has been first imported from **threex.proceduralcity.js** (https://github.com/jeromeetienne/threex.proceduralcity) and then some of them have been modified by us:

- Ground: a flat huge cube under player and objects;
- Sidewalks: flat cubes under buildings and lamps that, all together, form a single mesh. We rescaled and recolored each one;
- Lamps: each is made of three cubes, one for the base, one for the poll and the last for the head. In the head there has been placed a texture to simulate the light. We rescaled and recolored;
- Buildings: huge cubes on whose faces there has been placed the same texture, a pattern of white and black rows.

Other than these, since the repository is from 2014, it was not compatible with the latest versions like r105, so the code has been fully updated in order to make it possible to use.

The camera always looks ahead in front of the player and we chose to use the '*PerspectiveCamera*' since it mimics the way the human eye sees and hence matches perfectly the first person shooter games.

## 6. Lights and materials

The basic idea was to place a light of type '*PointLight*' (a light scattered from a point towards all directions) for each lamp. But since the number of lamps was 1600, this huge number of lights would be too heavy for the project. We therefore decided to follow another way: let lamps turn on around the player in a fixed radius and turn off the ones that were already turn on as soon as the player moved. Unfortunately, this way also turned out to be computationally heavy and we hence adopted just one light that worked the same way as

the sun: *'HemisphereLight'*, a light source positioned directly above the scene with color fading from the sky color to the ground color.

Since the *'MeshBasicMaterial'* is not affected by lights, for most of the object present in the scene we used the *'MeshLambertMaterial'*, a material for non-shiny surfaces without specular highlights. The shading is calculated using the *'Gourad'* shading model, which calculates shading per vertex and interpolates the results over the polygon's faces. Another important reason is the performance: due to the simplicity of the reflectance and illumination models, performances are greater than *'MeshPhongMaterial', 'MeshStandardMaterial'* and *'MeshPhysicalMaterial'* at the cost of some graphical accuracy.

## 7. Collision

One of the most difficult things to implement, along with the zombies, were the collisions. In a first phase of the project we used the *'Raycaster'*, a class mainly used for mouse picking (working out what objects in the 3D space the mouse is over) amongst other things. The *raycaster* simply provide a ray casted from an origin towards a direction and check if the ray hits something: if so, an array of sorted objects (distance metric) is returned and actions can be taken. We hence used an array of 16 rays casted towards the cardinal direction: North, East, West, South, North-East etc. In the render loop this method was computationally heavy and therefore we adopted another way: bounding boxes all over the object present in the scene. Indeed, *three.js* provides a boolean method named *'intersectsBox'* that check if two boxes intersect themselves. Collision are done as follows:

- Player and buildings: since the sidewalks were part of a unique geometry, we took each sidewalk separately in order to have a different bounding box for each sidewalk. Considering that the method *'intersectsBox'* works only between two bounding boxes, all around the player feet we placed a transparent box that always follows the camera to allow the function to check collisions;
- Player and map bounds: to deny the player from leaving the map we decided to put four invisible walls;
- Bullets and zombies: since the camera is fixed and it is not allowed to move it up and down, the collision between bullets and zombies can only occurs at the height of the camera. So for what concerns the orange zombies the collision is considered with the zombie's head, for mid height Hulk zombies with its body, and for giant zombies with both its lower legs.

Whenever a collision occurs, the player goes back into his old position. In fact, the player position is updated only if there is no collision, otherwise his current position is set to his previous one, which is always stored in a variable named *"previous_position"*.

## 8. Zombies as hierarchical models

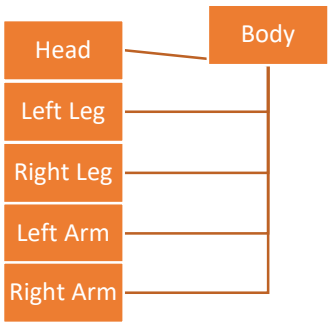
Hierarchical structures have been used to make the enemies in this game. Since the enemies are humanoids, the main advantage is the motions of the body, where each rotation or transformation of the parent elements affect also their children. Zombies are made of cubes, linked together to make hierarchical models. The orange zombie is the simplest one and is made of 5 elements: body, head, left and right legs, left and right arms. All the elements are linked to the body. The Hulk zombie is a taller zombie that is made of 9 elements: body, chest, head, upper and bottom left and right legs, left and right arms. All the elements are linked to the body, except for the bottom legs that are linked to the upper legs. The Giant zombie is made of 9 elements as well: body, head, upper and bottom left and right legs, left and right arms, a cube. The main element is an invisible cube at the bottom between the legs, used for the animations, linked to it there the body element, then everything is linked in the same way as the Hulk zombie.

The Animations have been created in the '*animate()*' function of the JavaScript through rotations and translations of the single elements of the zombies. Each zombie uses a '*lookAt()*' function to always look towards the player (camera) and walk in his direction. The *'lookAt()'* is used on the body element of the orange and Hulk zombie, but for the Giant zombie is used on the invisible cube because due to the height of the zombie its body would have tilted towards the player, therefore it has been necessary to use that cube positioned at its bottom. The orange zombie has a simple animation of arms moving up and down and legs forward and backward without elbows or knees. Instead both the Hulk and the Giant zombies have knees for a better visual effect since the camera is positioned at legs height and a more fluid walking motion is noticeable.

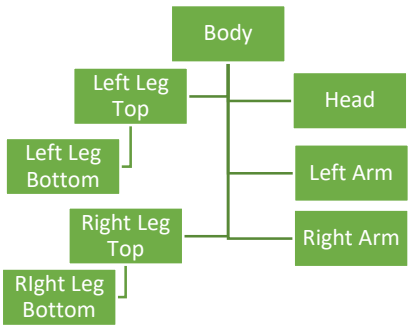
Since the zombie models are inspired by the Minecraft zombies, the starting zombie textures were taken from that game and then modified. Each face of each cube has its texture, so there are 6 textures for each element of the model. The textures of the Hulk zombie have been drawn from scratch.

Here are shown the three zombies hierarchical structures:

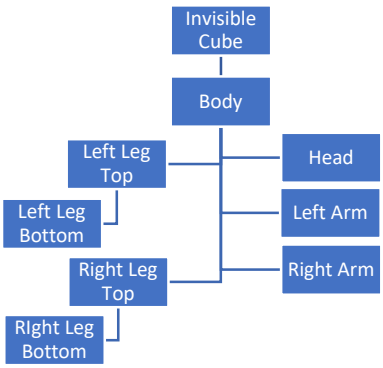
*Orange Zombie*




*Hulk Zombie*




*Giant Zombie*



## 9. Conclusion

The aim of the project was to make a browser game that meets some requirements. The project contains three hierarchical models representing zombies, each one with its peculiarities and its specific animations that included arm motion and fluid walking animations. The zombies were inspired from Minecraft zombies but all the textures used have been edited manually to perfectly fit our models. The lights are perfectly visible on the faces of the buildings in the city, showing a clear direction of the red sunlight and a motion sensor effect in the Dark mode that shows the lamps turning on and off based on the player's position. All these things concur to give to the user the idea of an apocalypse city where he has control over the zombies using his gun, being able to walk, run, shoot and aim like you would do in real life, avoiding of course the buildings due to collisions.