Final Project Report

Interactive Graphics
Giuseppe Capaldi, 1699498
Davide Lo Iudice, 1691799

Saturday 21st September, 2019

1 Abstract

The project idea is the creation of a First-Person-Shooter game in which the player can choose between three weapons with different features displayed after clicking on them, like the fire rate or the mag size, then by pressing start he is positioned in a map, generated with static and random elements. Using mouse and keyboard the player have to defeat the zombies that spawn incrementally in each wave. The game interface shows the life indicator, the bullets and the current wave number as a simple HUD.
The commands are:

- W,A,S,D = Move
- SPACE = Jump
- MOVING THE MOUSE = Look Around
- CLICK SX = Shoot
- CLICK DX = Aim
- R = Reload
- ENTER= shows map elements and zombies hitboxes

2 Environment: Three js

As 3D modeling environment we started trying using Babylon (this can be seen inside the repository branch: "testing3js"), but then we chose to move to a less videogame specific but somewhat more documented environment which is Three js. Three.js is the world's most popular JavaScript framework for displaying 3D content on the web, making possible to display models, create games, music videos, scientific and data visualizations, running on browser in mobile or desktop environments.
On three.js web page `https://threejs.org/examples/` can be seen some three js examples code and relative visualization. They generally inspired our work but none of them was taken specifically.

3 Libraries

The following are the libraries we used along with Three js APIs to build our videogame code.

3.1 Cannon js

Cannon js is a lightweight and simple 3D physics engine for the web. At `https://schteppe.github.io/cannon.js/` is possible to find some simple examples of physics based games or demos that show how powerful and simple is to use APIs of this library to handle collision and physics in general. At `https://schteppe.github.io/cannon.js/docs/` is possible to find a small documentation about these APIs. In the project Cannon js is used to handle the collision physics between walls and player, projectiles and environment, movement of the non playable and playable characters.

3.2 jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. In the project jQuery has been used to handle HUD animations (appearing disappearing of elements) and html related dynamic actions in general. This was the result of the choice to develop a three js independent GUI improving user interaction and usability of the game in general. Most used functions are: fadeIn() and fadeOut().

4 Technical

As first thing we can look into init() function called once window is loaded. This function includes :

- prescene(): above mentioned;
- initLoading(): initializes models and environment;
- initSounds(): initializes sounds;
- htmlInit(): initializes html elements;
- jQueryInit(): calls jquery functions;

Along with init() a initCannon() function is called where we initialize cannon world (similar for some aspects to a physical scene), all cannon related global variables and gives a body to the already created ground mesh.

4.1 Prescene

As first screen the user will see in the canvas three weapons' meshes rotating (incrementing their rotation property around the y axis every tick of animation) and translating towards the camera when they are selected (incrementing and decrementing their position along z and x axis). We want to point out that this (”prescene”) is a completely different scene object from the main game one (”scene”) that in the meantime is loading but it is not yet animated and visible. A simple ambient light is used in this scene. The description texts and the button are html/css elements animated with jquery, making them disappear when start button is pressed. A javascript onclick event will trigger a function that start the game, letting animation loop starts and making the main scene visible.

4.2 Environment loading

In initLoading() function we create the main scene, the main camera, the skybox and game map as a plane mesh with a repeated texture. Then we

initialize trough Cannon the physic body relative to the player character (not visible). To enable controls we istantiate the constructor PointerLock that is a class handling all player's controls and movements, and that has been customized to enable physics along with movements. Lights are a simple ambient light to see clearly all the environment similar to the one in the prescene and a pointLight to improve objects shadows visualization. In bitMaps.js file we created matrices as arrays of integer arrays that contain integer values specifing the type of object and the indexes represent its position in the environment, wallMaps does this for the objects and zombieMap for the zombies position. Some objects are randomly placed inside this matrix at start, others like the walls perimeter are prefixed. In scenario we initialized a dictionary with all the imported models path (obj and mtl files) and Three js created boxes with their textures (the ones with the property "internal" set at true value). These are meshes created using Three js geometry to load a personal texture instead of a predefined mtl file based material. It is used in a for loop where each different object is loaded as a new mesh put inside the relative property in the dictionary as a reference. Then, once all models are loaded, we call the function allResourcesLoaded() that is based upon the matrix wallMaps that according to the matrix, clones each type of mesh already loaded and after changing position adds it to the scene. This avoids to load the same model several times. CreateBoundCube() creates a custom physical body that is a box which measures depends from the ones of the mesh, making it always contained inside the box. Each mesh (internal or external) has an associated body created by the function createBoundCube() that takes the dimensions of the passed meshes with box3() and box3Helper() and along with these it creates a cube of similar size and a new mesh that identifies the body. Pressing ENTER key is possible to see a wireframe mesh representing this physical body. Bodies are created thanks to Cannon library.
InitSounds() function is used to load the music and the gun sounds. Three js offers a listener for events and a loader to load the different audio tracks. Listeners are used each one to control the audio sound or music singularly, to stop or play it in the proper moment.

4.3 Game loop: animate

Animate function is the real "heart" of the game, is a loop dynamically updating variables then its rendering the scene. SetAnimationLoop() in WebGLRenderer sets up the animation loop for us calling the function

animate.
So in animate() we update at runtime position of meshes making animations possible. For zombies each time animate is called and noZombie is true, a double for loop is run and depending from the index in zombieMap and the current waves value, places zombies for the next starting round . Then we update meshes positions according to the bodies positions. In animate we also check if player life is below 0 value, if so, game over screen can be displayed and the game stops, ready to be restarted (as a reload of the page). To remove sprites used to display life of zombies we call a function that acts as a garbage collector removing from the scene objects that are placed inside an array.

4.4 Zombie

Once zombies are placed, they follow the position of the user rotating towards the camera and moving in its direction. While doing this a movement animation later explained is performed. In each wave the zombies moves towards the camera with increasing speed, to be precise the speed increment of a multiplicative factor of:

$$(1 + zombieWave/5)$$

in this way for each wave the game becomes harder. Zombies have a damage property that is the aoumnt of player life they decrement when colliding with the player body and a life decrementing when they colllide with the bullets (when they are shot). When their life value is negative walking animation is stopped and their angular damping is set to zero, so when shot their body can roll. To avoid cloning problems related to GLTF objects and to avoid loading repetitions we chose to move zombies out of the playable area when they are considered to be dead. Then when the new wave starts they are moved again in the playable area and start again moving. Zombies current life can be seen from the player through a colored bar loaded as a sprite object with a png image as texture. When the life value changes the current sprite is removed from the scene and a new one with the image related to the decremented life is added.

4.5 Animation

All animations are made from ourselves without any pre-loaded animations. They are based on rotation positions or translation positions of meshes. For zombies we rotate legs, arms and torso alternatively while they are moving towards the player, simulating a zombie like walking animation. When the zombies life value is negative or zero the animation stops and their arms are raised up through a rotation. For guns we moved them changing their position and rotation when a reloading occurs or when right click is pressed or released to make a ”aim-like” animation. Also bullet moving is an animation possible using velocity property related to cannon library acting like a force impulse when they are fired.

4.6 Shooting dynamic

We have a different shooting effect depending on the selected weapon, what changes is the mass of the bullet along with the velocity of the bullet and the damage that is caused to the zombies when colliding with their bodies. There are two types of fire mode Auto and semi auto. The first one let the player hold mouse down and this will countinuosly fire a bullet at a defined rateo (depending on the selected gun) thanks to a interval event listener. The second one will just fire a bullet for each click if not waiting for the next bullet.

5 User Interaction

So the user interactions are the following: the user can choose the weapon according to the difficulty that derives from using it, then with W,A,S,D keys player moves in the scene and can jump with SPACEBAR. Player goal is to stay alive as mush as possible avoiding contact with zombies bodies. To complete the round player has to shoot to every zombie present in that round until their lives reach zero values and disappear. The player can use right click to use a more precise aiming, for the sniper is present a zoom and a scope image. Then the player has to reload using R key when there are no more ammonition or whenever the mag is not full.

References

[1] Edward Angel and Dave Shreiner, Seventh Edition code examples, `https://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SEVENTH_EDITION/CODE/`

[2] Three js library. `https://github.com/mrdoob/three.js/`

[3] Cannon js library. `https://schteppe.github.io/cannon.js/`

[4] Examples on jquery and documentation. `https://www.w3schools.com/jquery/`

[5] Website for most of the free-license models. `https://www.kenney.nl/`

[6] Textures and hud images. `https://opengameart.org/`

[7] Sounds to be free downloaded. `https://patrickdearteaga.com/arcade-music/`

[8] Sounds to be free downloaded. `http://soundbible.com/tags-gun.html`