FINAL PROJECT

FLAVIO GALASSO 2034377

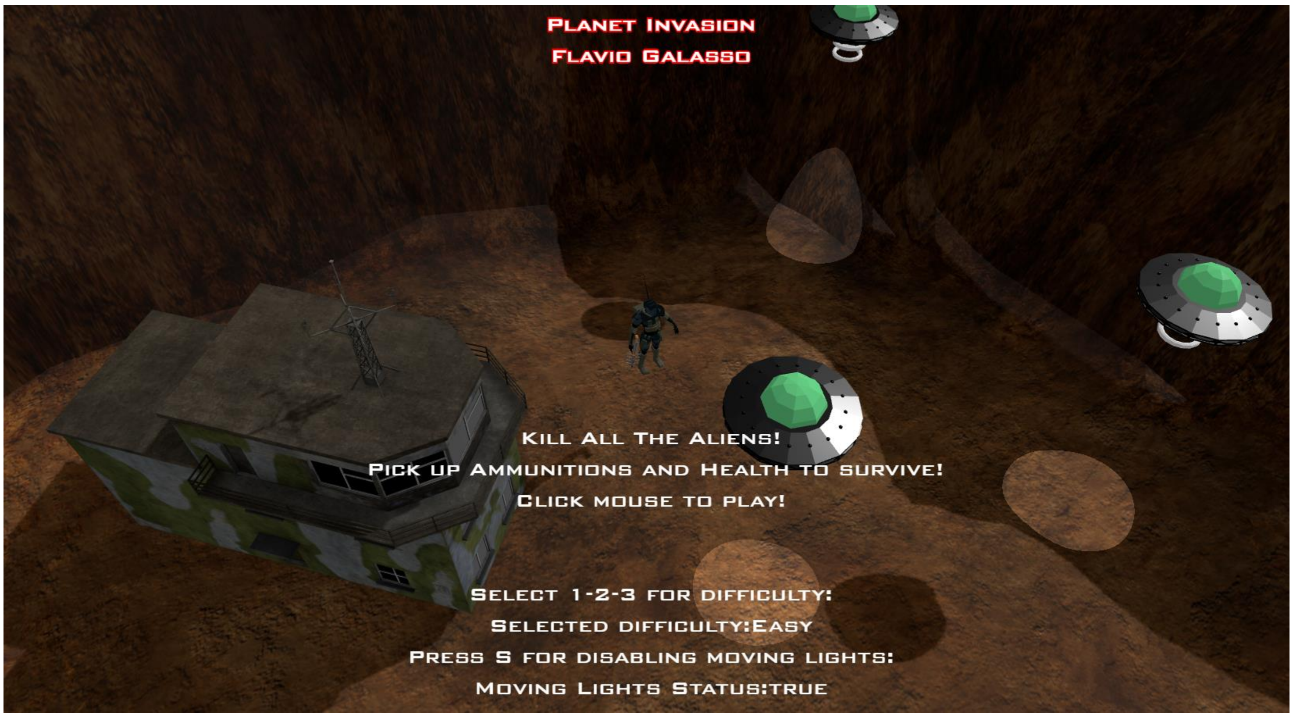
16 July 2022

Contents:

1. Introduction

1.1. Idea

The idea behind this project is to build a mini-game set on a planet where there is a human base to defend against an alien invasion. The protagonist, a player-controlled robot, must withstand the waves of increasing difficulty by dodging shooting aliens and dodging their bullets. The player has a limited number of ammunition and health points at his disposal and, during the gameplay, it can recharge them by collecting the appropriate pickups.
The game is arcade-style, so it has an infinite gameplay with increasing difficulty as long as the player can stay alive.



Planet Invasion
Flavio Galasso
Kill All The Aliens!
Pick up Ammunitions and Health to survive!
Click mouse to play!
Select 1-2-3 for difficulty:
Selected difficulty:Easy
Press S for disabling moving lights:
Moving Lights Status:true

1.2. Project Requirements

The project must include:

- Hierarchical models:
  2 Hierarchical models were used: Alien and Robot Models
- Lights And Textures:
  Multiple statically and dynamically spawned lights and textures attached to the 3d models were used.
- User Interaction:
  The user can control the movement, rotation and shooting direction of the Model.
- Animations (Not Imported)
  Multiple animations were manually interpolated using Finite state machines for the models and eased with the TWEEN Library.

1.3. Libreries Used and Requirements

- Threejs 0.142.0
- Express 4.18.1
- TWEEN.js (https://cdnjs.cloudflare.com/ajax/libs/tween.js/16.3.5/Tween.min.js)
- Tested on Chrome - Version 103.0.5060.114 (Official Build) (64-bit)
- Windows 10 20H2
- Nodejs LTS 16.16.0x64

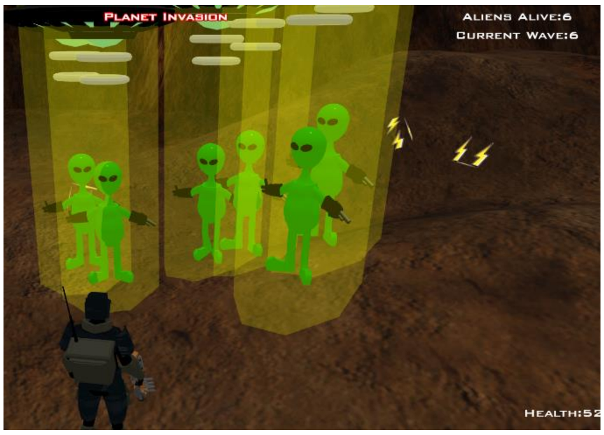
1.4. Model links

- Alien: https://www.turbosquid.com/3d-models/free-alien-character-3d-model/533251#
- Robot: https://www.cgtrader.com/free-3d-models/character/sci-fi-character/low-poly-military-robot
- Weapon: https://www.turbosquid.com/it/3d-models/weapon-cyborg-model-1592582
- Terrain: https://www.turbosquid.com/it/3d-models/terrain-multiplayer-3ds-free/667801
- Building: https://www.turbosquid.com/it/3d-models/free-airbase-control-tower-3d-model/934801
- Ufo: https://www.cgtrader.com/free-3d-models/aircraft/other/low-poly-ufo-6c16e512-00db-441c-87cd-61ea644707ee
- Ammo Image: https://www.flaticon.com/free-icon/bullets_224678
- Health Image: https://www.flaticon.com/free-icon/flash_252590
- SkyBox Textures: RIPPED FROM Call Of Duty 4,https://www.customapscod.com/mod.php?id=NDE=

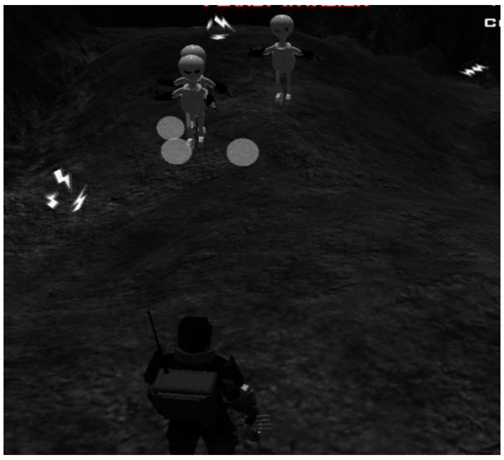
1.5. Game Execution

1. Clone the repository
2. Extract the zip folder
3. Inside the folder: npm install
4. To start the game: npm start
5. With Chrome, navigate in the link provided by the terminal

1.6. Game Manual



Once the game window is opened, the player finds himself in an introductory screen where the map, the main character and some introductory writings can be seen. Here the difficulty can be selected with the 1-2-3 keys and the game can be started with the click of the left mouse button.
Throughout the game there will be waves of aliens to fight, the waves will be endless and of incremental difficulty, and the ultimate goal is to survive as many waves as possible.



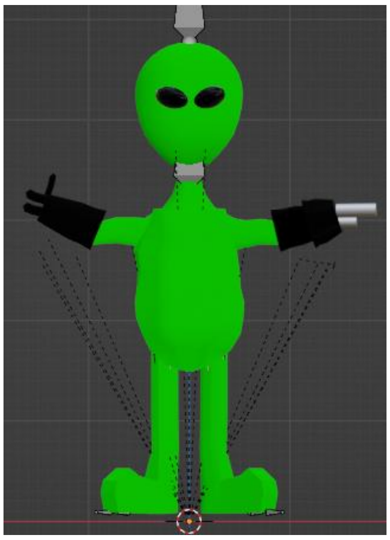
To move the character, the player must use the WASD keys, to jump and avoid bullets the SPACE key, to aim just place the mouse pointer on the alien to be shot and make a single click. In the inferior part of the screen some indicators for life and ammunition can be seen, and the player must pay attention to these values because if life goes to zero the game is lost, if the ammunition goes to zero it can no longer shoot. To recover or collect life or ammo, the player has to walk across the map, dodge bullets and collect pickups the pickups. When the life or ammunitions go under a critical level, the colors of the game change to highlight the risky situation, putting more emphasis to the bullets and the powerups on the map.

2. Model Implementation

2.1. Robot Model

This model was downloaded for free from the internet and then modified to change the pose and add the weapon, also downloaded and imported and added to the skeleton on the right hand. The final model was developed with blender and exported in GLTF format, ready to be loaded natively in THREE.js. This model packs a lot of different textures and materials, and most of them were supported natively by THREE.js.

2.2. Alien Model



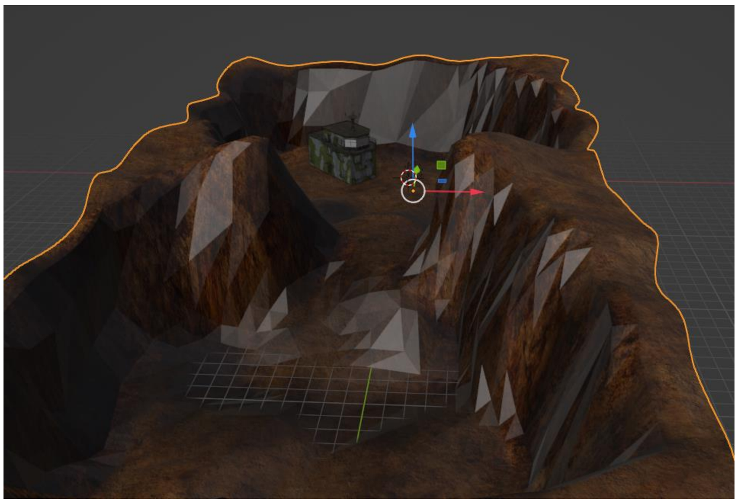
This model was downloaded for free from the internet, but it was not complete at all. It was necessary to reshape the model, use new textures and build a completely new skeleton. This required a lot of manual work and it has been done in Blender and Mixamo.

Once the final model has been completed, it was exported in GLTF format, ready to be loaded natively in THREE.js.

2.3. Other models

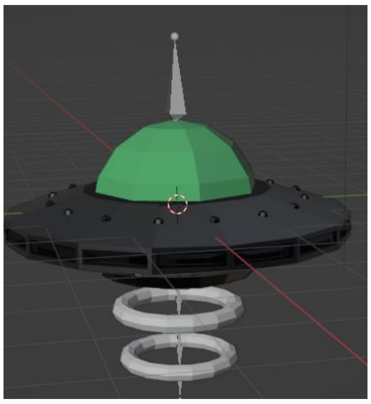
Other models have been downloaded, reshaped and exported to gltf, among these we find:

- Planet Map with Tower Building

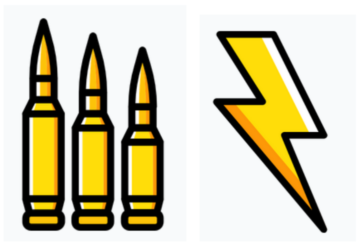


Originally the tower was not present, in facts it has been imported from another model, placed, sized and re-textured accordingly.

- Ufo Model



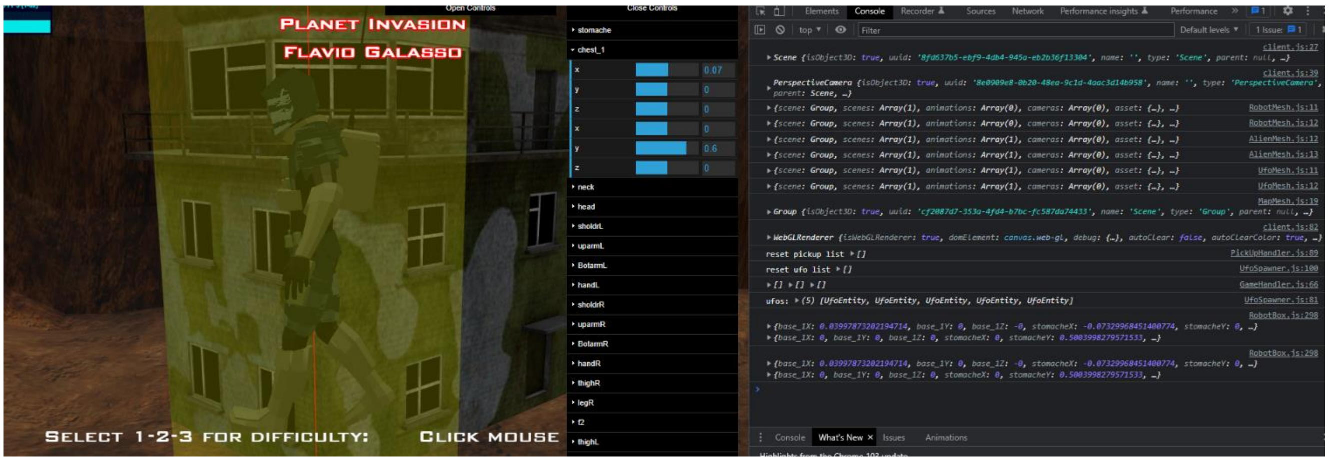
2.4. Ammo/Health Pickups



Cubes have been created for the pickups and assigned transparent textures created from scratch on threejs from images taken from the internet.

3. Animations

Since one of the project's requests was to create animations from scratch without importing them, a debug version of the game was created precisely to vary the position and rotation of the bones and export keyframes such as to import them into the actual game with just one copy and paste.



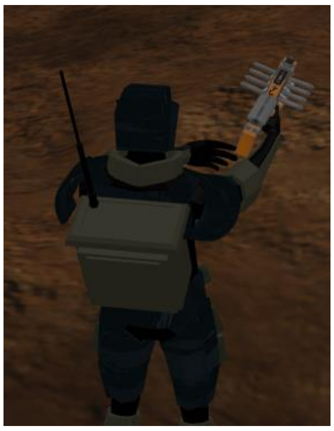
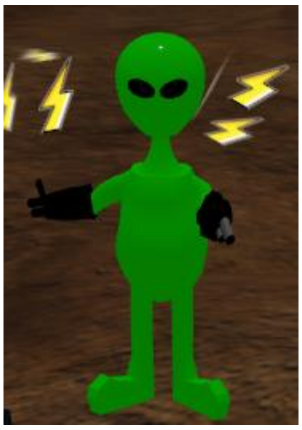
```
                                                                                      client.js:27
▸Scene {isObject3D: true, uuid: '8fd637b5-ebf9-4db4-945a-eb2b36f13304', name: '', type: 'Scene', parent: null, …}
                                                                                      client.js:39
▸PerspectiveCamera {isObject3D: true, uuid: '8e0909e8-0b20-48ea-9c1d-4aac3d14b958', name: '', type: 'PerspectiveCamera',
 parent: Scene, …}
▸{scene: Group, scenes: Array(1), animations: Array(0), cameras: Array(0), asset: {…}, …}    RobotMesh.js:11
▸{scene: Group, scenes: Array(1), animations: Array(0), cameras: Array(0), asset: {…}, …}    RobotMesh.js:12
▸{scene: Group, scenes: Array(1), animations: Array(1), cameras: Array(0), asset: {…}, …}    AlienMesh.js:12
▸{scene: Group, scenes: Array(1), animations: Array(1), cameras: Array(0), asset: {…}, …}    AlienMesh.js:13
▸{scene: Group, scenes: Array(1), animations: Array(1), cameras: Array(0), asset: {…}, …}      UfoMesh.js:11
▸{scene: Group, scenes: Array(1), animations: Array(1), cameras: Array(0), asset: {…}, …}      UfoMesh.js:12
                                                                                      MapMesh.js:19
▸Group {isObject3D: true, uuid: 'cf2087d7-353a-4fd4-b7bc-fc587da74433', name: 'Scene', type: 'Group', parent: null, …}
                                                                                      client.js:82
▸WebGLRenderer {isWebGLRenderer: true, domElement: canvas.web-gl, debug: {…}, autoClear: false, autoClearColor: true, …}
reset pickup list ▸[]                                                              PickUpHandler.js:89
reset ufo list ▸[]                                                                  UfoSpawner.js:100
▸[] ▸[] ▸[]                                                                          GameHandler.js:66
ufos: ▸(5) [UfoEntity, UfoEntity, UfoEntity, UfoEntity, UfoEntity]                    UfoSpawner.js:81
                                                                                     RobotBox.js:298
▸{base_1X: 0.03997873202194714, base_1Y: 0, base_1Z: -0, stomacheX: -0.07329968451400774, stomacheY: 0, …}
▸{base_1X: 0, base_1Y: 0, base_1Z: 0, stomacheX: 0, stomacheY: 0.5003998279571533, …}
                                                                                     RobotBox.js:298
▸{base_1X: 0.03997873202194714, base_1Y: 0, base_1Z: -0, stomacheX: -0.07329968451400774, stomacheY: 0, …}
▸{base_1X: 0, base_1Y: 0, base_1Z: 0, stomacheX: 0, stomacheY: 0.5003998279571533, …}
>
```

This idea was immensely helpful in facilitating the creation of complex animations using visual feedback.

3.1. Idle animation

The idle animations are simple, smooth, and do not provide confusion with the rest of the animations. For the Alien, it will slightly rotate the head, for the Robot, it will slightly rotate the chest as it is seeking for enemies.

3.2 Shooting animation

3.3 Jumping animation

This animation was a bit tricky because the skeleton was not really able to flex enough for my idea of a jumping animation. The end result, with TWEEN.js helping greatly, was a small springiness added to the legs and the upward motion of the arms. The physics actually wait for the first stage to end before jumping. Also no other jumping commands are allowed during the duration of the animation.

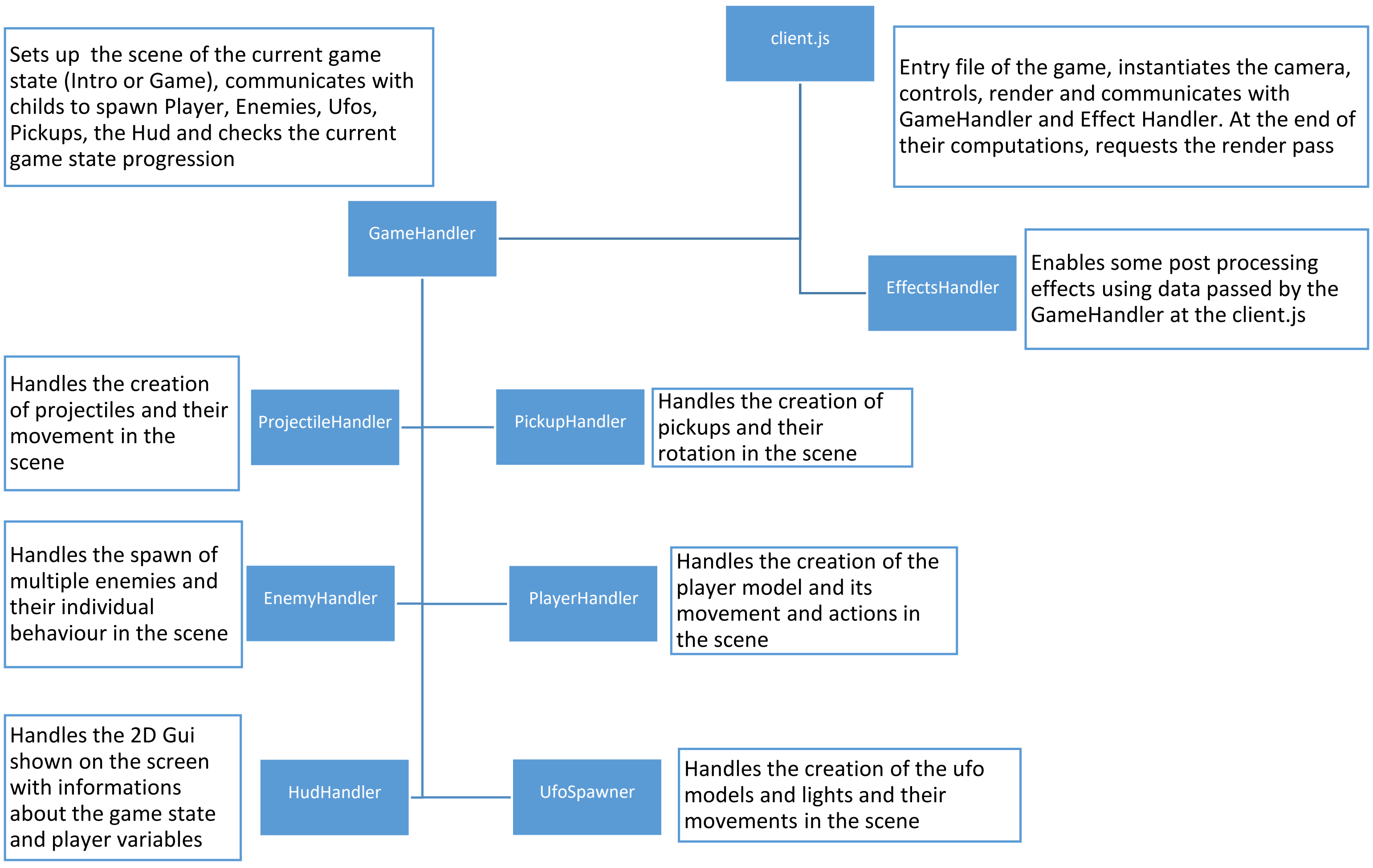
3.4 Walking animation

The alien model has only 2 bone configurations, while the robot model has 4 for better quality.

4 Functionality implementations

The game code has been implemented using an object oriented logic: entities have been created for each single purpose, among these we find:

- GameHandler
- EffectsHandler
- PickupHandler
- ProjectileHandler
- EnemyHandler
- PlayerHandler
- UfoSpawner
- HudHandler

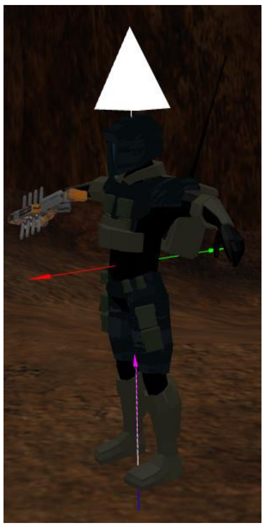

client.js
Entry file of the game, instantiates the camera, controls, render and communicates with GameHandler and Effect Handler. At the end of their computations, requests the render pass
Sets up the scene of the current game state (Intro or Game), communicates with childs to spawn Player, Enemies, Ufos, Pickups, the Hud and checks the current game state progression
GameHandler
EffectsHandler
Enables some post processing effects using data passed by the GameHandler at the client.js
Handles the creation of projectiles and their movement in the scene
ProjectileHandler
PickupHandler
Handles the creation of pickups and their rotation in the scene
Handles the spawn of multiple enemies and their individual behaviour in the scene
EnemyHandler
PlayerHandler
Handles the creation of the player model and its movement and actions in the scene
Handles the 2D Gui shown on the screen with informations about the game state and player variables
HudHandler
UfoSpawner
Handles the creation of the ufo models and lights and their movements in the scene

4.1 Robot Box

Both PlayerHandler and EnemyHandler share the same basic structure: RobotBox.js.
The purpose of this component is to:

- create a copy of the model previously loaded in RAM (robot or alien)
- manage all status variables such as Health or Ammo
- manage the finite state machine to define the animation to play
- play animations
- handle movement and rotation, gravity
- manage raycasts for collision detection
- many other things

It also contains the keyframes data for both models, used for interpolating bones rotations using TWEEN.js.
One particular aspect to talk about is the management of raycasts. In fact, 5 raycasts have been created:



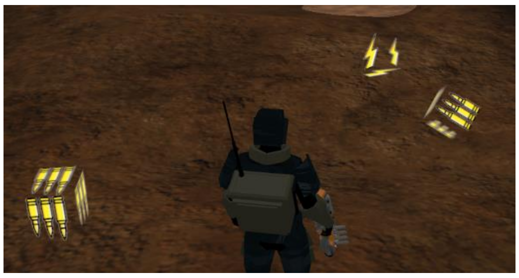
- Front and rear (red and green) to manage the collision with the map
- Lower and Upper (blue and violet) to manage gravity and the "bump up" effect if the model lands just below the ground
- Lower directed towards the head and very long (white) to manage the impacts with the bullets

All those raycasts were setup to work correctly with both models. To reduce computational power, their intersections are calculated every 50ms.

4.2 PlayerHandler

The PlayerHandler module extends the RobotBox module, and takes care of controlling the movement and actions of the RobotBox through the user input.

4.3 PickupHandler



Handles the spawn of individual pickup objects that can be picked up by the PlayerEntity. The number of total generated boxes is limited to reserve computational power and balance the game. The position and type of the pickup objects is randomly generated around the player.

4.4 Shooting system

The shooting is handled by the ProjectileHandler entity that handles the shooting requests by the EnemyEntities and PlayerEntity, spawns and updates bullet positions and directions. The bullet models are different if shot by the player or by the aliens. After some time, if the bullets do not hit anything, they get removed from the game to save computational power.

4.5 Health/Ammo System

The health and ammo status variables are kept in memory by the corresponding RobotBox entity assigned to the single EnemyEntity or PlayerEntity. Health can go beyond 100 and it helps to survive longer in the most difficult modes. Ammo is limited only in the PlayerEntity, and does not allow firing when it goes below or equal to zero.

4.6 Game Watchdog System

The GameHandler entity is responsible for handling all the above mentioned entities, controlling and adjusting the progress of the game. If the player dies, it initiates a procedure of resetting all the variables, destroying all the generated models and building the introduction scene, ready for a new game. During the game, it spawns and moves ufos and directional lights at random positions by the UfoSpawner entity and it also communicates with the EffectsHandler to apply a post-processing darkening effect for when the player heath or ammunitions are below a critical threshold.

4.7 Lighting System And Materials

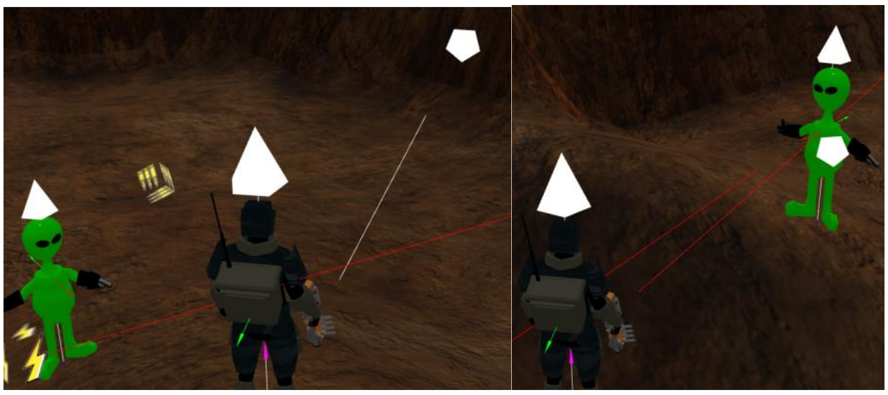
Multiple lights were used all over the scene:

- Soft Ambient Light
- Directional light: used to simulate a "sun", pointing from far and at angle so all the models could cast their shadows on the terrain and receive other shadows.
- Ufo Spotlight: moving and dynamic lights attached to every ufo, used to improve the user immersion in the game (turned out to be very difficult for low-end CPUs to handle, hence they can be disabled)

As Described in the models chapter, all the models have been assigned with different type of textures. Multiple types have been tried: MeshPhongMaterials, MeshLambertMaterials and finally MeshStandardMaterials. The last ones were choosen because they were very intuitive to tweak by Roughness and Metalness, and accurate enough for the project.

5 User Interaction

This program is very user interactive, in fact the player can control his model using the classic WASD movement keys, jump with the SPACE key, aim with the mouse and shoot with a click. To facilitate the aiming, the GameHandler has an "aimbot" feature, that guarantees perfect aim for the enemies to the player (always) and for the player to the enemies (but only if the mouse is hovering the target)



During the introduction scene, the player can select the difficulty with 1-2-3 keys, and this will directly affect both the enemies' and player's bullet damage:

1. Easy mode: very difficult to die and enemies will die with 1 shot
2. Medium mode: very difficult to die and enemies will die with 2 shots
3. Hard mode: very easy to die and enemies will die with 2 shots

Since the game is not able to hit 60 Fps reliably on all machines, the player can also toggle on/off the dynamically generated and moved lights from the ufos, always from the introduction scene.

6 Final Considerations

Programming this game was the most difficult programming challenge of my life (circa 2568 lines of code), in fact I had to learn how to use Blender to modify the models and I found a lot of obstacles that I had to overcome, between them the most troublesome were:

- If each spawned entity loaded the .gltf model, the program presented significant framerate drops, so all the models were loaded only once at the beginning of the program and then cloned via ram for each entity. The "object3D.clone()" function of threejs does not work with skinned meshes (skeleton) so a custom function has been made.
- The generation of dynamic lights (for example during the spawn of the alien when it is released by a ufo) created drop in framerates, the solution was to create a fake light with a cylinder and a semi-transparent yellow texture.
- Getting the models to stay inside the arena while still being physics-consistent using raycasts was very difficult, especially the part to make the models follow the shapes of the map (bumps) in a natural way. Also a nice function has been implemented: if the player or the enemies fall out of the map, they will die instantly. This discourages the player to go out of bounds and as a bonus also kills enemies that spawn in wrong places, freezing the game progression.
- The aiming system was one of the major let-downs of this project, in fact over the course of months many methods of aiming and shooting have been tested. The best result would have been by also rotating the model's chest according to the aimed target, but the threejs "object3d.lookat()" function does not work well for models with skeletons. Furthermore, the choice of the z plane during the 2d -> 3d conversion made the aim unpredictable, in fact a raycast was added to identify the correct z of the entity the player really wanted to shoot. The raycast search was limited to enemy meshes only to reduce computational power, so shooting at the map instead of the enemies can be inaccurate.
- Lots of other minor optimizations were used for aiming at a steady framerate of 60fps, and all the physics behind movements and rotations are scaled to work in the same way with different framerates. Sadly, not all machines I've tested it on can run it at 60fps, some more complicated CPU bound optimization should be done. The game is also resistant to window resizing in every part of the game, in facts it is able to adjust its proportions on-the-fly.