Interactive Graphics
Final Project report

GIASCA team

September 25, 2022

Academic Year:	2021-22
Partners:	Antonello Giorgio, 1836529
	Cecilia Assolito, 1857897
	Andrea Cadoli, 1837028

GitHub Repository:
https://github.com/SapienzaInteractiveGraphicsCourse/final-project-giasca-team
Demo Website:
https://sapienzainteractivegraphicscourse.github.io/final-project-giasca-team/

Abstract

Our Final Project for the Interactive Graphics course is a single-player web-game called **STRANGER KILLS**: its name is a play on words between 'Stranger Things' (a popular science fiction horror Netflix Series, on which our game is based) and 'Kills' (the goal of the game).
The aesthetics of the website is based on the recurring themes in Stranger Things, while the main goal of the game is to kill all the monsters that chase your character through all the map, trying to hurt it as much as they can.

Contents

1 Environment

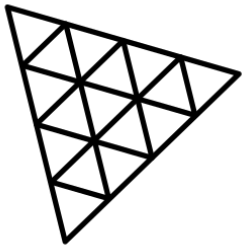
Our project was developed using Visual Studio Code connected with the GitHub repository, in order to exchange updates between all the members of the team through different branches.
In the **main branch**, we uploaded the final version of our project, which is a web-game based on HTML/CSS/JavaScript code.
To render the scene and all the models in it, we used **Three.js**, while, to manage the rules of the world that we've built, we have exploited the physical properties and features provided by the **Cannon.js** physics engine.
Last but not least, we used simple JavaScript programming to move the different hierarchical models in the project.
We've tested our web-game only on the Chrome browser.

2 Libraries, tools and models

2.1 Libraries

- ***WebGL***: It is a JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser.

- ***Three.js***: Three.js is an open-source library that is based on WebGL.
  We used it in order to implement **scene**, **cameras**, **objects**, **materials**, **lights**, **camera movement**, with OrbitControls, and **GLTFLoader** in order to load models and textures.



- ***Cannon.js***: It is an open-source JavaScript 3D physics engine that supports different kind of shapes such as box, plane, cylinder.

2.2 Tools

- ***GoogleFonts***: is a library of open source font families and APIs for convenient use via CSS.
  We used it to implement better looking writings in the gameover/win screens.

2.3 Models

We have imported all the models (**without the built-in animations**) from **Sketchfab**: is a 3D modeling platform website to publish, share, discover, buy and sell 3D content. It provides a viewer based on the WebGL technologies that allows users to display 3D models on the web.
Sketchfab relies on the WebGL JavaScript API to display 3D on web pages in all major modern web browsers. It does not rely on third-party plugins.
The hierarchical models used in this project are:

- ***Female Officer (Trinity in the game)***: is one of the two main characters that can be chosen from the options in the home of the website. We've implemented three moves for it: walk, run and punch.

- ***Eleven***: is the other one main character. We've also implemented three moves for it: walk, run and a handful (which in the series has mysterious powers attached to it).

- ***Vecna***: is the monster that chases your character. We've implemented two moves for it: walk and slap.

We have also loaded models for user interactions:

- ***Vecna's Clock***: if the character finds and hits it, the fog in the map will be removed, easing the view for the character.

- ***LightBulb***: if the character finds and hits it, the street lamps put in the map will be turned on, easing the view for the character in the main corners of the map.

The other models (benches, car, street lamps, tree forest, tree trunk, wood fence, wood house) are simply loaded in the map once the game is started to build the environment of the game.

3 Technical aspects

3.1 Features

3.1.1 Map

The map is consisted of a thin box as a stage, 1 house, 4 lamps, a park with 3 benches ad fences, 1 destroyed car, 1 forest, 1 tree trunk, 1 clock, 1 lightbulb and (it depends if day or night is selected) 1 sphere representing the sun or 1 sphere representing a blood moon.

3.1.2 Camera

There are 2 cameras, one used as a 3rd person camera and the other is an adjustable one. The 3rd person camera will always follow the character's position with a default initial distance. The adjustable camera will make the user choose how to move and zoom in/out(it will change the default initial distance) around the map, but it can not go lower than 0 on the y axis in order to always see the elements. In order to not make the two cameras go in conflict, a fake camera was introduced as a clone of the 3rd person one.

3.1.3 Lights

There are in total 9 lights: 1 ambient light, 5 spot lights(4 of them to emulate the lamps light and 1), 2 directional lights, 1 hemisphere light All of them are in fixed position except for the spotlight linked to the center of the moon, the light moves as the moon moves and targets perpendicularly on the ground.

3.1.4 Fog

: There is a greyish/sky blue THREE.js fog and its density depends on the difficulty chosen.

3.1.5 Textures

When the game starts, we load all the textures individually. There are 2 cube textures (one for the day and one for the night), 1 bump texture used for the stage and 2 color textures for sun and moon.

3.2 Management of the character-world interaction

3.2.1 The *Finite State Machine*

At first, we have to describe the way in which the movements of both the characters and the monsters are built and managed.
The concept at the base is the *Finite State Machine*, which in general is defined as a *mathematical model of computation*. It is an abstract machine that can be in exactly one of a finite number of states at any given time, which, in our case are: Idle, Walk and Run. The FSM can change from one state to another in response to the keyboard inputs, in particular:

- W/S: walk
- SHIFT + W/SHIFT + S: run
- else: Idle (do nothing)

This structure is very useful for us because it permits to create several states through which we can easily manage the different movements created for the character/monster.

Moreover, in our FSM we have a dictionary (called "meshesDictionary") which contains the references to the different parts/joints of the model (obtained by the .gltf file in the specific model folder): the references of the hierarchical model in the .gltf allow us to change the rotation of all the body parts without calling a function that gets the particular part ("getObjectByName") every time.
So, in each state (one class for each state) we can set what we want the model to do by simply taking the values from meshesDictionary.

3.2.2 The Controllers and the using of Cannon.js

To move the hierarchical models in the map, we used the *controllers*, which are built in order to take the inputs (by the keyboard) and set the proper values for translation and rotation vectors of the model, and to update the state of the FSM they refer to. We associated every loaded model to a Cannon.js body and shape: this resulted very useful, especially for the collisions, since Cannon.js has its own collision-avoidance algorithms. The only Cannon bodies with a mass greater than zero are assigned to the main character and to the monsters; the other ones are static bodies, therefore they cannot be moved.
Every character/monster has 2 bodies, one sphere at its feet used for the movements on the map and one box at their shoulder/back, that always follows the sphere movement on the x and z axis, used to detect collisions between the punch and monsters: in fact, when the character receives the input command to punch, the box-shaped body goes forward with respect to the character.
So it simulates the movement that the arm does to give the punch, and if the body collides with a cannon body that belongs to a monster, then that monster is killed.
Obviously, if the character tries to go further than the borders of the map, he is considered as dead, so his cannon body and his mesh are removed from the map and the game ends with a "GAME OVER". The same removal happens with the bodies of the monsters.

3.3 Relevant Functions

- ***insegui_meglio***: this function takes as input a monster's and character's meshes and bodies. Once the monster spawns, the function will rotate the monster in order to look always at the character and make it follows the character with a certain velocity(it depends on the difficulty chosen).
- ***check_borders***: checks if the character has fallen outside the perimeter of the map and gives a message of error/dying.

3.4 Collisions

We have 4 possible collisions:

a. Monster/Character spheres bodies – Models bodies

b. Monster sphere body – Character sphere body environment.

c. Character box body – Monster box body

d. Character box body – Clock/Lightbulb model body

In case a) nothing special happens, the models' bodies remain still because they are setted as static bodies.
In case b) it means that a Monster reached the Character, the collision will increment a variable conta_collisioni and if it matches the variable fine_gioco, the game

ends.
In case c) the punch reached the monster and it will make it disappear/die, also a variable quanti_fuori, it stands for the total kills, will increments.
In case d) the punch reached the one of these models and the interaction's result will happen.

4 Implemented interactions

- It's possible to turn on the 2 directional lights and the lamp lights if the player punches the model of the lightbulb.
- It's possible to make the fog go away by punching the model of the clock.
- The user can select the difficulty of the game, it will affect the velocity and total number of the monsters, the fog density and the maximum of the hits the main character can take before dying.
- The user can choose the main character
- The user can choose the daytime (night or day)

5 Further implementations and improvements

- ***Monsters that give Slaps to the character***: the function is already implemented in the project, we have already modeled all the animations to do it, but it doesn't work very well
- ***Home responsive***
- ***Possibility to mute/unmute sounds***
- ***Possibility to use weapons***: if the character finds and hits it, the fog in the map will be removed, easing the view for the character.
- ***More interactions with the environment***: like a mystery box that gives you a weapons or the possibility to enter in the house
- ***More animations***: like death of the monster/character