Paint-It!

Table of contents:

Paint-It! 1
- Environment and Libraries 2
  - Client 2
  - Server 2
  - Tools 2
- Architecture 3
- Technical Aspects 4
  - Map 4
  - Model of Mech 4
  - Animations 4
  - Movements and Collisions 4
- User Guide 5
- Reference 6
- Concatcs 6

Paint-It! is a first person shooting game developed with Three.js. This is a multiplayer game 1-VS-1. It consists in painting the platforms present in the arena and then killing the enemy.

Environment and Libraries

Client

The project is developed in *JavaScript* language using the libraries Three.js and Physijs for modeling 3D and Socket.IO for the communication with the server.

- **Three.js** is a library used to create and display animated 3D computer graphics in a web browser. Three.js uses WebGL.
- **Physijs** is a three.js framework that implements collision of 3D objects.
- **Socket.IO** a Node.js framework for communication between client and server

I use Three.js because it allows you to manage WebGL at a higher level and create easier complex structures.

To detect the bullet collisions, I use Physijs because the high speed of the bullets required complex calculations.

Server

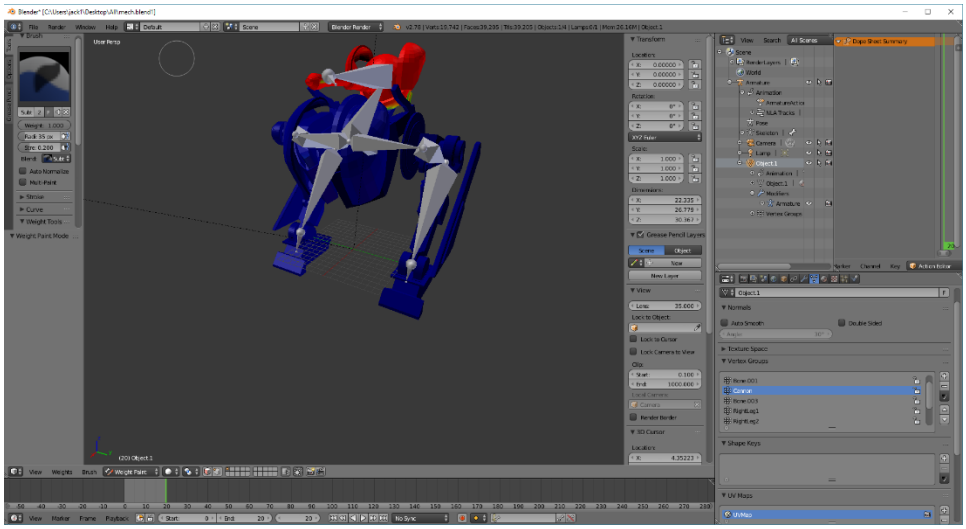
The server is in *Node.js*, a run-time environment for executing JavaScript code server-side. The server uses 3000 as port-number.

The main page of the site is in *HTML* and *CSS* and the libraries used are:

- **jQuery** to navigate the pages of the site.
- **Socket.IO** for communication client-serve

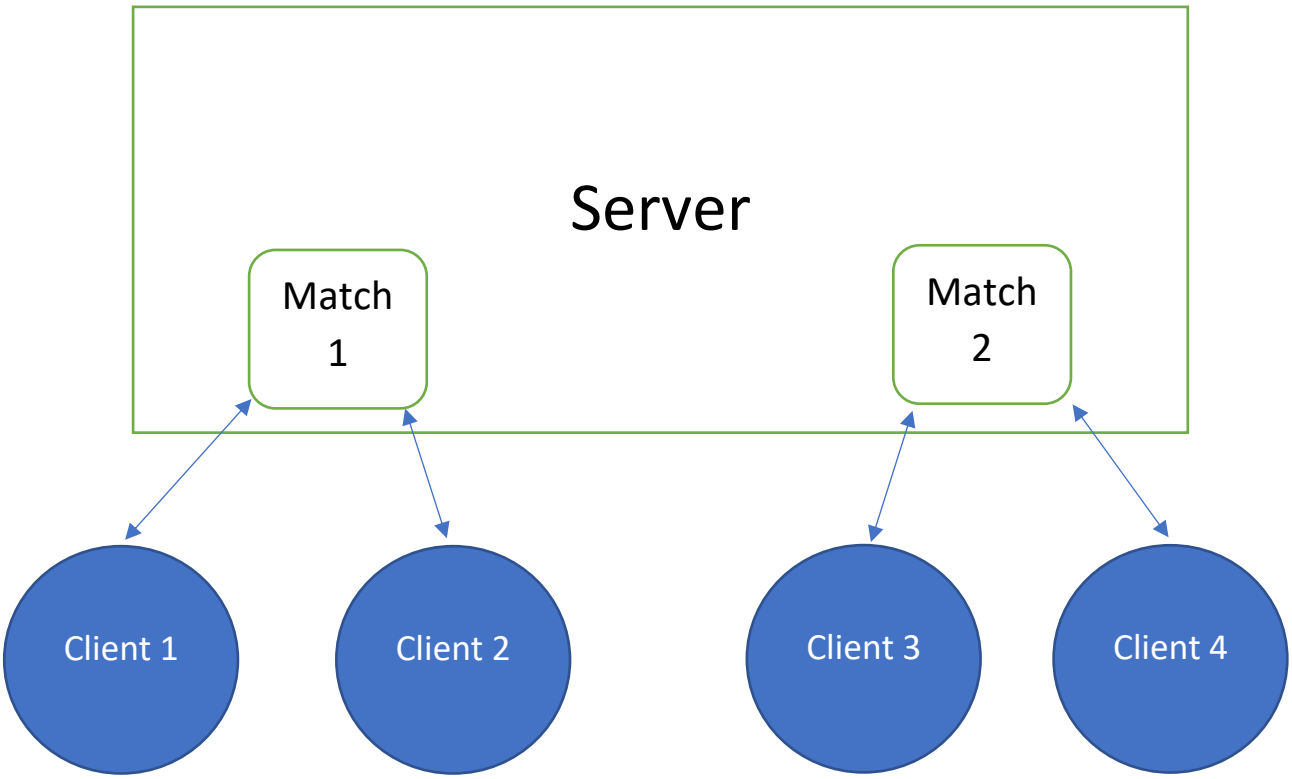
Tools

I used **Blender** to modify the model of the Mech and create a skeleton. Then with "**Three.js Blender Export**" (extension for Blender found on GitHub) I exported the project in *json* format to use it in my project. (Other details in Technical Aspects chapter).



Architecture

The server creates a match when two clients connect to the server and with the match making combines two user. The communication is real-time.



The server has a list of all connected sockets, each socket is in a different state:

- **Connected**: when a client is in menu (index page).
- **Pending**: when a client choose a username and waits for an enemy.
- **In Game**: during the match.

When a socket disconnects from the server, it is removed from the list.

When start the match the server prepares the necessary informations (position of platforms, position of player 1 and player 2) and sends this data to the clients.

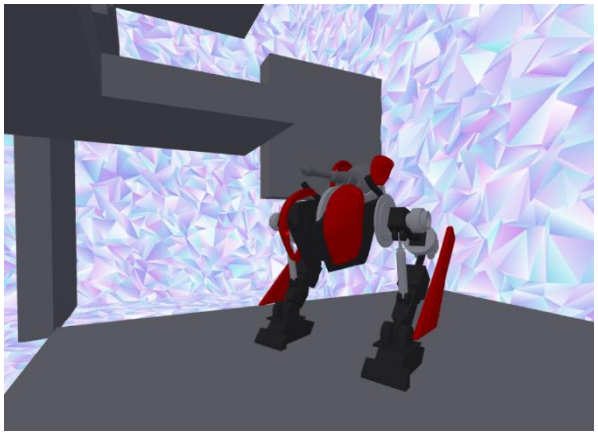
During the match a client sends the data to server that re-sends them to the second client.

Technical Aspects

Map

The map is a cubic arena, in which there are grey platforms. The position and the rotation of this platforms are random and decides on the server. Then it sends the data to the two clients. A futuristic texture was applied in the grounds.

Model of Mech



I found the model of Mech online and I used *Blender (open source modelling program)* for create a skeleton of the model and modify something. I colored the model, I created the skeleton of the model and I set the weights of vertices for each bone. Then the model is exported in json for Three.js using an extension of Blender.

Animations

The animations of the “walk” and of the “idle” are have been realized with *Blender* then they are exported in *json* format to be used in Three.js as animation. The Mech can move also his cannon (up and down) when the aim changes. When the Mech runs the “walk” animation is faster; if the gravity changes, the model is overturned.

Movements and Collisions

I realized the collisions of the player using the *raycaster*, a particular Three.js object to detect the collision, so the player can move and jump in correct way.

The movement follows the physical laws of gravity of the real life, so the speed and the position are calculated with math formulas. In this game there is the possibility to change the direction of the gravity (the changing is independent between two player). To implement this feature the sign of gravity is inverted. The change of severity does not affect the opponent.

For the bullet collisions the *raycaster* doesn’t work so I used the library *Physijs*. The bullets indeed are too fast, so it must use a complex calculation to detect an eventual collision. When a bullet hits something an event is emitted.

When a player does any movement sends this data to the enemy, passing through the server.

User Guide

The user must choose a username and wait for an opponent.

When starts the match there is a countdown of 5 seconds.

In the arena there are 80 grey platforms, they are disposed and rotated in random way, the server decides the disposition.

The controls of movement are standard:

- W: move forward
- A: move left
- S: move backward
- D: move right
- Shift: (pressed) run
- Space: jump

In this game it possible change the direction of the gravity, upward and downward. The changing is independent between two players. For change the gravity the commands are:

- E: upward gravity
- Q: downward gravity

With the left click of mouse you can shoot. If your bullet hit a platform you paint the platform blue, if your enemy hits a platform with a bullet it paints the platform red. The goal of the game is to paint as many platforms as possible of own color (blue).

Each player has 3 lives, when a player is hit by an enemy bullet he loses a life.

The match finish when one of the two loses all lives, but the winner is the player that has paint more platforms.

Reference

- Three.js documentation: https://threejs.org/docs/
- Blender: https://www.blender.org/
- Three.js Blender Exporter: https://github.com/mrdoob/three.js/tree/master/utils/exporters/blender
- Stackoverflow.com: https://stackoverflow.com/questions/11473755/how-to-detect-collision-in-three-js
- Three.js example: https://threejs.org/examples/misc_controls_pointerlock.html

Contacts

Project Link: https://github.com/jacopo1395/Paint-It

Email: mailto:jack1395@hotmail.it