Tanks Commanders

Reinforcement Learning Project

Itay Lavi-Okabi

# Introduction

## Project Design

### System Design

The system is divided to “back” – meaning the ML algorithms & communication, and “front” – a GUI for visualizing the tanks to the client. The client should be able to visualize the algorithm as he sees fit, which is why the back is completely separated from the implementation of the GUI, as long as it follows the API.

The client holds a GUI and connects to the back. In response he gets a finished gameplay – text describing the game. The gameplay includes initializing and positioning the tanks, every command given to the tanks, and the game’s aftermath.

### Game Flow & Algorithmic Plan

A **turn** – one frame. Hence the unit measure is [F]. Assume 25 frames per second [FPS].

Controlling the tanks

A tank can do only one of the following actions in each turn:

1. Move forwards/backwards (change position).
2. Turn by x degrees.
3. Turn its turret by x degrees.
4. Shoot. Shooting creates a bullet in the environment, and its movement is one sided and constant in velocity. Once the bullet reaches the shooter’s max range/hits the target it’s removed from the game.

Battle Zone (Reinforcement),

Training Zone (Genetic)

Each tank decides what to do every turn – moving, shooting, or scanning. Once a decision is made (let us call it a **plan**) a list of proper steps for executing it is generated, containing the following options (let us call them **actions**): moving one step forwards/backwards, rotate, rotate turret or shoot. Remember: the step size or rotating angle a tank can make in a single turn is determined by its properties. So, if a plan is saying “rotate 90” and the tank is slow – say rotation size is 0.5 – it will take it 180 turns to execute this plan (around 90F / 25FPS = 7.2 Seconds).

#### Genetic Algorithm

Bigger body – Higher Defense, Lower Mobility

Bigger turret – Higher power, Slower to rotate

Higher Cost – Larger Sonar & Better Reliability, Lower Chance to being produce

## Plan

We start with a basic GUI made with React.js that allows to place, move and rotate Tank objects. After that, we go to building the back – a Node.js server that will cover the Python ML algorithms as mentioned above. Then it’s time to work on what we all gathered here for – an amazing top notch edge of technology Machine Learning. To start off easy, we will actually begin the ML journey with the Genetic Algorithm that will – hopefully – improve the tanks used

# GUI & Tank control (React.js)

Focusing on visualizing and controlling the environment and its objects.

31/08/2022 –

1. Creating an Environment & Training Zone
2. Creating Tank object that can be placed in different positions.

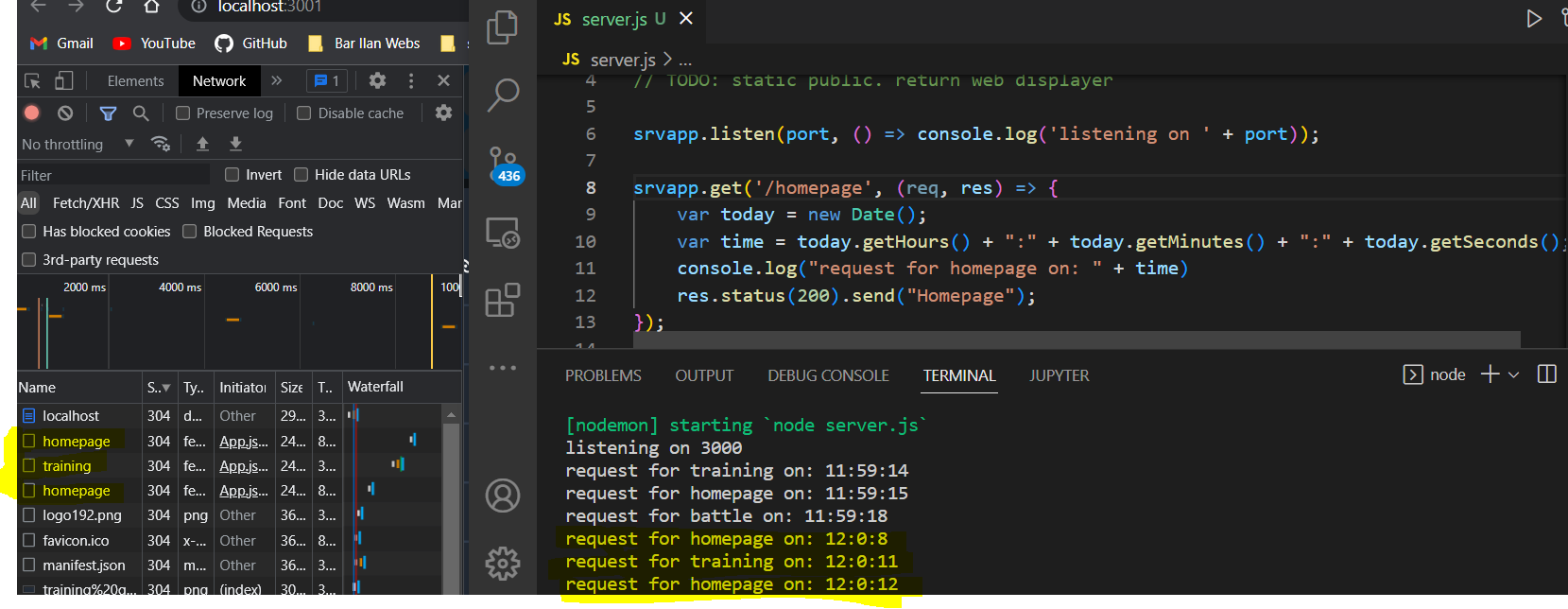
A screenshot of a computer

Description automatically generated with low confidence

## Controlling the game – (Node.js server)

### Creating server

1. Create a node.js server – also install Express.
2. Send & receive message on client side.
3. Create a simple RESTful API for receiving homepage (empty for now), testing request, battle request.



1. Send a temporary file (representing a valid game file) and parse it.

Notable issues at this point:

* New subjects in JS: express, cors, spawn
* API problems (fixed when URL in fetch changed to “http:\\localhost\api” instead of “\api”).
* Parsing the file on the client – had hard time comparing the lines to string “g” (it marks the end of the tanks initialization, and the start of the game itself. Problem fixed by using trim() on the lines read from file).
* Running a python script from the server. A python script is supposed to create the game file and of course run the game itself. So, I already started here to create the python class “BattleGenerator” to handle creating and managing the battle file. Then created two JS modules – API (to contain all actions regarding the communication and API requests) and python\_manager (to contain all methods for handling python scripts).