HFLT - Hyper Frictionless Laser Tanks

COS426 Final Project - Abdur-Raheem Idowu - PLAY HERE

Abstract

Fast-paced, first-person, multiplayer game. You control a high tech, diamond plated, laser-equipped, frictionless tank! But so do your friends (read: enemies). Who will come on top? Easy to learn but hard to master. Implemented with a **broadcasting server model** and hosted on Google Cloud Compute. Game map generated procedurally and naturally with **Poisson Disc Sampling**. Heads up display elements through a **HTML overlay**.

Goal

To make a game that would be infinitely fun to play with friends. Aimed for simplicity, polish and replayability instead of intricacy. I knew I wanted to make a multiplayer game so I could put my Websocket experience to good use, and I knew it would be more fun if it was competitive. Many related games already exist (FPSes) so there was lots of precedent and resources!

Methodology

For **multiplayer**, as explained above, I used Socket.IO. The server's job was only to relay positions/rotations and events such as “playerShot” back and forth, trusting each client completely. This has the pro of being extremely fast and I have yet to experience game-affecting latency. It works best when the game won’t be released to the general public. However, it breaks when players tab out of the game tab, because then the client isn’t processing logic and is frozen with respect to other players. I did fix the most obvious manifestation of this issue, i.e. when you shoot a tabbed out player, by hiding their character until they respond to that event. The implementation consists of several handlers for different client/server messages on both sides to accomplish this. A folded view of the handlers in the server side code is below:

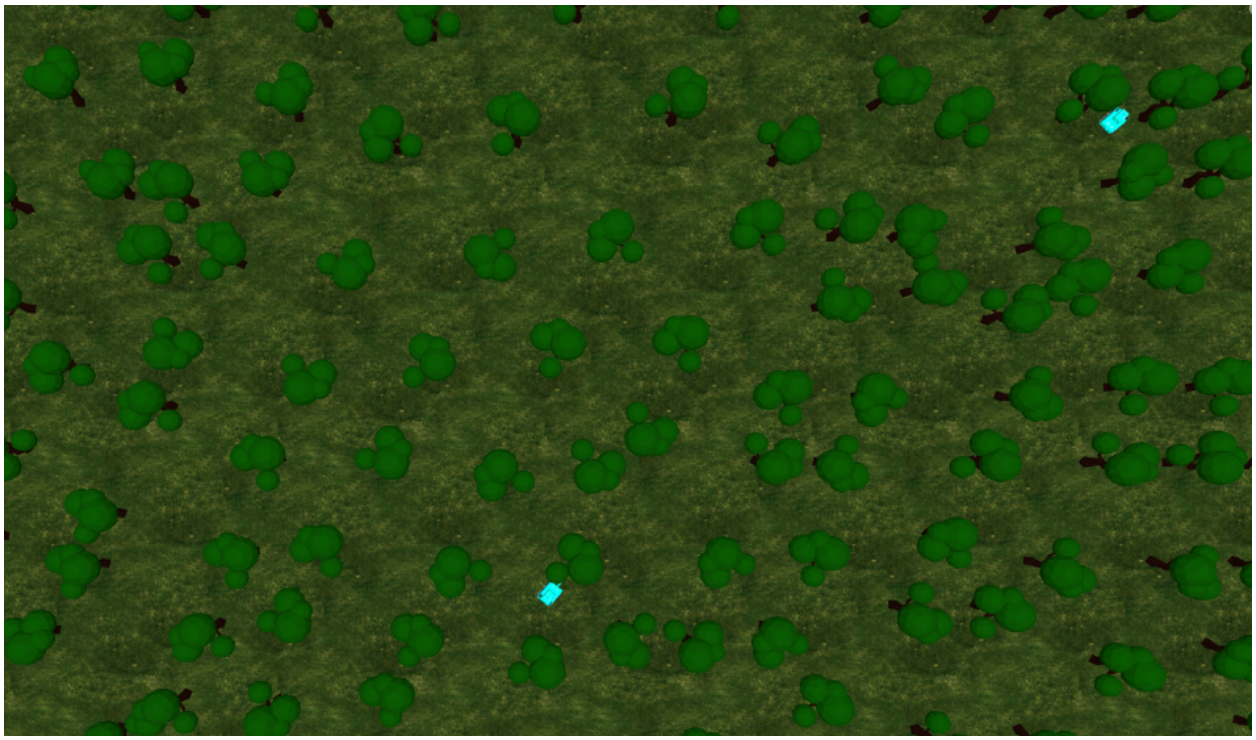
```
socket.on('clientUpdate', (data) => {…
});

socket.on('PlayerShooting', () => {…
});

socket.on('disconnect', () => {…
});

socket.on('PlayerShot', (name) => {…
});
```

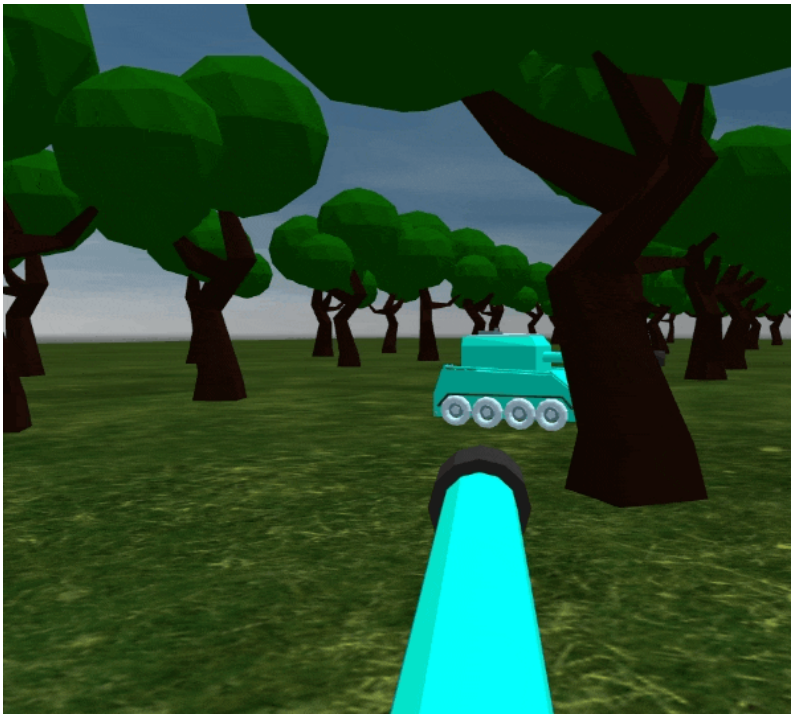
For **map generation**, I wanted to keep the map simple, but still wanted it to offer some gameplay consequences / possibilities. I settled with trees that block visibility, hinder movement and laser shots, allowing skilled players to use them effectively. (Please ignore the fact that in my game world lasers can destroy laser tanks but don't penetrate trees 😂) I used Poisson Disc Sampling, implemented and run once on the server side, to generate the map. Here is a top down view of the map (with two tanks ready for intense combat). Skybox done with THREE.js.



I am happy with this procedural generation method because when weaving through the trees it really feels like a real forest.

For **player-tree collisions**, I used a self-coded sphere-sphere model. The tank has an irregular shape and I wanted to support both rotation and lateral movement (frictionless tanks), so I thought anything more complicated wouldn't be worth the effort. Player-player collisions are not implemented because before players come that close, they probably would have already eliminated each other.

For **raycasting**, I used the built in THREE.Raycaster. I got a list of intersections sorted by distance, and used that to allow trees to block lasers. See GIF on the next page. Even though the trees are thin, it's really fun to try and psyche your enemies and dodge their attacks. This gameplay is facilitated by a two second laser cooldown, making it crucial to time your shots. (Quick aside, I implemented mouse rotation controls but since I wanted this game to be playable in lecture when you don't have access to a mouse, I went with arrow keys for rotation instead.)



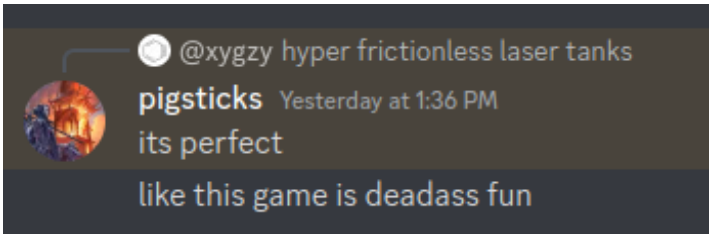
(boom💥)

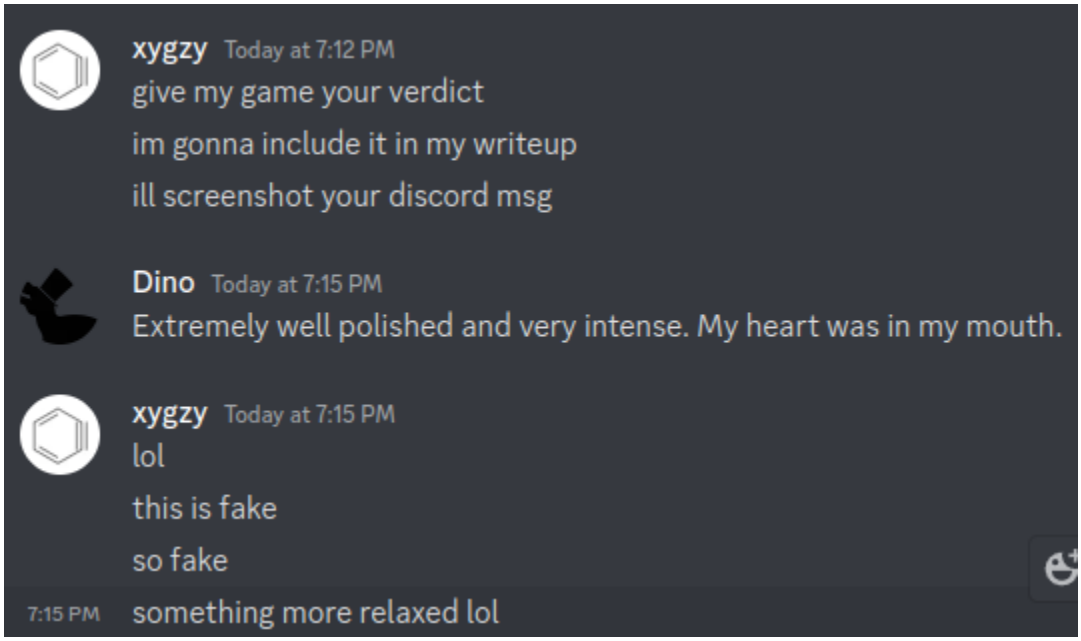
To communicate with the player when their laser was ready, I used a HTML overlay as a **HUD**. I already had experience with DOM manipulation with Javascript, so this was straightforward. To increase visibility of the elements against the trees, I used a partially blurred background with CSS, reminiscent of tech-tech fighter jet HUDs.

I was thinking of letting players choose their own name in real .io game style, but I didn't want to deal with the logic. There's the pro that if I wanted to release this game to the public I wouldn't have to worry about inappropriate names. However, it does take away the fun of playing with > 2 friends because it is hard to keep track of who is who.

Results

I had some friends try it out - their (unbiased) testimonials speak for themselves on the next page.I believe that if the game is fun, I did a good job. I think I'll continue playing and improving on this game with my friends and siblings. Powerups, more varied environments/tanks, etc; the simple gameplay framework means that the possibilities are limitless.





(we had an intense match)

Discussion and conclusion

My initial idea was "Project Maze Racers". I pivoted away because for a maze to be fun to navigate, it needs to look relatively varied so you can locate yourself / have a sense of progress as you delve into it. I didn't think I could find high quality varied assets so I switched it up. Very satisfied with that choice. Indeed, Maze Racers would only be truly fun at a high cooperation and competitive level, so I doubt it would have the same pick-up and play element as HFLT.

I've always wanted to make a quake style, edge of your seat, single shot elimination game. While this game doesn't provide the verticality Quake-like shooters offer (because I didn't want to deal with physics), I think it comes close in terms of fluidity, action, and most importantly, fun!

Works cited:

Tree model - https://poly.pizza/m/MSuchZNT2G
Tank model - https://poly.pizza/m/5rqAPFRwLMh
Poisson Disc Sampling Tutorial - https://sighahttps://opengameart.org/content/30-grass-textures-tilableck.com/post/poisson-disk-sampling-bridsons-algorithm
Ground texture - https://opengameart.org/content/30-grass-textures-tilable
Skybox textures - https://opengameart.org/content/retro-skyboxes-pack