VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

3D Multiplayer Shooting Game

By

Phan Đỉnh Thanh Hoàn

A report is submitted to
Lecturer: Assoc. Prof. Nguyen Van Sinh

Ho Chi Minh City, Vietnam
Year 2023
Scala

ACKNOWLEDGEMENTS

It is with sincere gratitude and profound appreciation that I express my acknowledgment of the invaluable guidance and unwavering support provided by Assoc. Prof. Nguyen Van Sinh. Throughout my journey, his continuous encouragement, expert advice, and mentorship have been instrumental in shaping my path towards achieving my aspirations.

His dedication to fostering growth and his exceptional commitment to my success have profoundly impacted my academic and personal development. I am truly indebted to Assoc. Prof. Nguyen Van Sinh for his invaluable mentorship, which has not only enriched my academic pursuits but has also instilled in me a sense of determination and resilience.

I am deeply thankful for his unwavering belief in my capabilities, which has empowered me to strive for excellence and reach new heights in my endeavors.

Note: Paper A4, Top: 2.5cm; Bottom: 2cm; Left: 3cm; Right: 2cm

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 : INTRODUCTION

1.1. Overview

Scala is a 3D multiplayer shooting game developed using React Three Fiber, ReactJS, and the Playroom kit. It combines the power of React Three Fiber, a framework for creating 3D applications in React, with ReactJS, a popular JavaScript library for building user interfaces. The utilization of the Playroom kit enables the game to incorporate multiplayer functionality, allowing multiple players to engage in real-time gameplay experiences.

Key Features:

- ❖ 3D Environment: Scala leverages React Three Fiber to render immersive 3D environments, enhancing the gaming experience with visually appealing graphics and effects.
- ❖ Multiplayer Functionality: Enabled by the Playroom kit, the game supports multiplayer interactions, allowing multiple users to engage in real-time gameplay, fostering competitive or cooperative experiences.
- ❖ Shooting Gameplay: The game focuses on shooting mechanics, providing players with various weapons and challenges within the 3D environment, emphasizing combat and strategy.
- ❖ ReactJS Interface: ReactJS ensures a smooth and interactive user interface, allowing players to navigate menus, access options, and interact with the game elements seamlessly.

1.2. Rules

The rule of the game is simple :

1. Enter the lobby.
2. Customize your profile. (user’s profile is auto-generated on lobby startup)
3. At this stage, you could either host the game by yourself or invite other people to join.
4. Enter the game.
5. When entering the game, you’re automatically chosen a character (or model).
6. Walk around the map, find other players and shoot them.
7. You score 1 kill-point for killing a player, or gain 1 death-point for every time you dies.

CHAPTER 2 : TOOLS

2.1. Overview

Tool	Description
Iterm2	A feature-rich terminal emulator for macOS, offering a highly customizable and powerful command-line experience. It provides numerous enhancements over the default macOS terminal, including split panes, multiple tabs, advanced search, hotkey customization, and support for a variety of shells.
Quaternius	A library of hundreds of free Low Poly 3D Models, using the CC0 License.
Tmux	A terminal multiplexer that allows users to create and manage multiple terminal sessions within a single window. It enables splitting, organizing, and managing different terminal panes and windows, facilitating efficient multitasking and workflow management for command-line users.
Neovim	A modern, highly extensible text editor based on the Vim text editor. It aims to improve upon Vim's features, performance, and extensibility while maintaining compatibility with Vim. Neovim offers enhanced plugin support, better integration with modern systems, and a more maintainable codebase for developers.
Fork	Fork is a Git client designed for macOS, Windows, and Linux, providing a user-friendly interface for working with Git repositories. It offers a visually appealing and intuitive way to manage branches, commits, merges, and other version control operations, making it easier for developers to interact with their Git repositories.
Github	A web-based platform for version control using Git. It provides a collaborative environment for software developers to host, review, and manage code repositories. GitHub offers features such as issue tracking, pull requests, wikis, and actions, facilitating collaboration among developers and enabling efficient software development workflows.

ReactJS	ReactJS is a JavaScript library for building web applications and user interfaces. It's like a magical toolbox that helps you create awesome and interactive websites. React lets you break down your web app into smaller components, making it easier to manage and update. It's like Lego blocks that you can put together to build a whole website.
Three.js	A popular JavaScript library used for creating and displaying 3D computer graphics in web browsers. It simplifies the process of working with WebGL, providing a high-level abstraction for 3D rendering, allowing developers to build impressive 3D visualizations and games on the web.
React Three Fiber	React Three Fiber is a React renderer for Three.js, offering a declarative way to create 3D scenes using React components. It bridges the gap between React and Three.js, allowing developers to build complex 3D applications using familiar React paradigms.
React Three Drei	A collection of useful helper components and hooks for React Three Fiber. It provides additional abstractions and utilities to simplify common tasks when working with React Three Fiber, enhancing the development experience for creating 3D applications in React.
React Three Rapier	A physics library built on top of React Three Fiber and Ammo.js. It enables developers to add realistic physics simulations, such as collisions, gravity, and constraints, to their React Three Fiber-based 3D scenes and applications.
Playroom Kit	A tool provides developers with a lightning-fast backend multiplayer toolkit for games. Build and scale games effortlessly, with zero server setup.
Blender	Blender is a versatile and powerful open-source 3D creation suite, offering a comprehensive set of tools for various aspects of 3D modeling, animation, rendering, simulation, and more. It's widely used by artists, designers, animators, and game developers due to its extensive capabilities and flexibility.

Table 1. Technologies used

2.2. Installation

2.2.1. Tmux

MacOS: You can install tmux using package managers like Homebrew.

1. Open Terminal.
2. Install Homebrew if you haven't already by running: /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
3. Install tmux via Homebrew: brew install tmux

Linux (Debian/Ubuntu): Install tmux via the package manager.

Windows: For Windows, you can use the Windows Subsystem for Linux (WSL) to install tmux.

2.2.2. Neovim

MacOS: brew install neovim

Linux (Debian/Ubuntu): sudo apt-get install neovim

Windows: Neovim can be installed via Chocolatey or by downloading the executable from the Neovim website.

2.2.3. Iterm2

iTerm2 is available on the App Store or can be downloaded directly from the official website.

2.2.4. Quaternius

Quaternius is not an installable software but a website/platform where you can download 3D models and assets created by the Quaternius user. Visit the Quaternius website and navigate to the models or assets section for downloading.

2.2.5. Fork

MacOS: Download the Fork app from the Fork website and follow the installation instructions.

Windows: Download the Fork app from the Fork website and follow the installation instructions.

2.2.6. Github

GitHub is a web-based platform accessible through a browser. Simply visit GitHub and sign up or log in to your account.

2.2.7. ReactJS

ReactJS is used within a web project and is added as a dependency. To set up a React project, you can use tools like create-react-app.

2.2.8. Three.js

Using npm or Yarn:

1. Navigate to your project directory in the terminal.
2. Run either of the following commands based on your package manager preference:
    - npm : npm install three
    - Yarn : yarn add three

2.2.9. React Three Fiber

If you haven't already set up a React project, create one using create-react-app or any other preferred method.

Within your React project directory, install React Three Fiber via npm or Yarn:

- npm : npm install @react-three/fiber
- Yarn : yarn add @react-three/fiber

2.2.10. React Three Drei

Ensure you have React Three Fiber installed in your project (as mentioned above).

Install React Three Drei within your React project:

- npm : npm install @react-three/drei
- Yarn : yarn add @react-three/drei

2.2.11. React Three Rapier

Make sure you have React Three Fiber already set up in your project (as previously instructed).

Install React Three Rapier in your React project:

- npm : npm install @react-three/rapier
- Yarn : yarn add @react-three/rapier

2.2.12. Playroom Kit

Install Playroom Kit can use npm or Yarn

- npm : npm install playroomkit
- Yarn : yarn add playroomkit

CHAPTER 3 : IMPLEMENTATION

3.1. Methods & Functions

3.1.1. Create player(s)

First, I create an array using React state to store players.

```
const [players, setPlayers] = useState([]);
```

Using Playroom Kit, I can easily handle room creation, joining and also let players pick their names, colors and avatars. Once host taps "Launch", your game will start.

```
// Show Playroom UI, let it handle players joining etc and wait for host to tap "Launch"
await insertCoin();

// Start the game! (your game code below)
startGame();
```

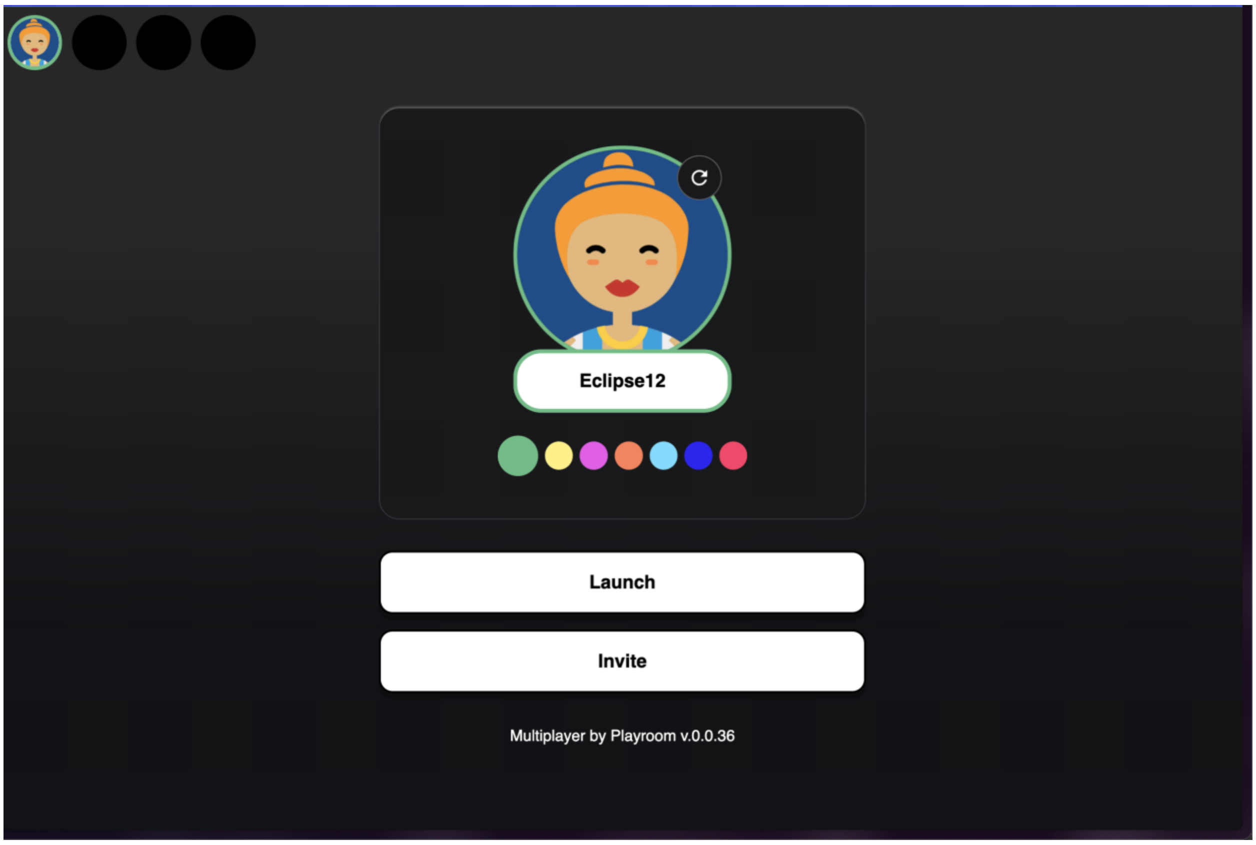

Figure 1. Game Lobby

Here is the full implementation of starting the game :

```
const start = async () => {

  await insertCoin();

  onPlayerJoin((state) => {
    const joystick = new Joystick(state, {
      type: "angular",
      buttons: [{ id: "fire", label: "Fire" }],
    });
    const newPlayer = { state, joystick };
    state.setState("health", 100);
    state.setState("deaths", 0);
    state.setState("kills", 0);
    state.setState("model_num", Math.floor(Math.random() * 4) + 1);
    setPlayers((players) => [...players, newPlayer]);
    state.onQuit(() => {
      setPlayers((players) => players.filter((p) => p.state.id !==
state.id));
    });
  });
};

```

Code Explanation :

❖ onPlayerJoin((state) => { ... });: This function listens for events triggered when a player joins the game. When a player joins, it executes the provided callback function, passing the player's state as an argument.

❖ Const joystick = new Joystick(state, { … });: For each joining player, a joystick controller is created based on their state. This joystick seems to be designed to create a user interface (UI) for the player to interact with the game, specifically for controlling movement or actions. It includes buttons, in this case, a “Fire” button, and may have additional functionality based on the specified type (“angular” in this instance).

❖ const newPlayer = { state, joystick };: A new player object is created, storing the player's state and the joystick instance associated with that player.

❖ Setting Initial State Values: Initial state values such as "health," "deaths," "kills," and "model_num" are set for the player upon joining the game.

❖ Adding and Removing Players: The new player object is added to the players array using the setPlayers function. Additionally, an event handler is set up to remove the player from the players array when they quit the game (state.onQuit()).

Finally, in order to implement the start() function, I implement the useEffect() in React to call the function once every the game renders.

```
useEffect(() => {
  start();
}, []);
```

3.1.2. Firing bullet(s)

First, we define an if statement when user is pressing the “Fire” button on the screen. If the user is pressing, set the animation of the character. And only if the user is hosting the game (in order to differentiate within other players), I need to detect the date of the last fire in order to ensure the delay between shots.

```
    if (joystick.isPressed("fire")) {
      // fire
      setAnimation(
        joystick.isJoystickPressed() && angle ? "Run_Shoot":"Idle_Shoot",
      );
      if (isHost()) {
        if (Date.now() - lastShoot.current > FIRE_RATE) {
          lastShoot.current = Date.now();
          const newBullet = {
            id: state.id + "-" + +new Date(),
            position: vec3(rigidbody.current.translation()),
            angle,
            player: state.id,
          };
          onFire(newBullet);
        }
      }
    }

```

Code Explanation :

❖ if (joystick.isPressed("fire")) { ... }: This condition checks if the "fire" button on the joystick or controller is being pressed.

❖ setAnimation(...): This function sets an animation based on certain conditions. If the joystick is pressed and an angle is present, the animation "Run_Shoot" is set; otherwise, the animation "Idle_Shoot" is set. This seems to control the animation displayed based on player actions.

❖ if (isHost()) { ... }: Checks if the current player is designated as the host. This condition might be used for handling specific actions that should only be performed by the host player.

- if (Date.now() - lastShoot.current > FIRE_RATE) { ... }: Checks if enough time has passed since the last shot (based on FIRE_RATE). It ensures a delay between consecutive shots to control the firing rate.

- Creating a newBullet object:
  - id: Combines the player's state ID with the current timestamp to generate a unique identifier for the bullet.
  - position: Determines the position of the bullet, likely by retrieving the translation from a rigidbody.
  - angle: Represents the direction or angle at which the bullet is fired.
  - player: Associates the bullet with the ID of the player who fired it.
  - onFire(newBullet);: Triggers an onFire event/function and passes the newBullet object as data. This could be a function responsible for handling bullet creation or firing events in the game.

In order to store the bullet(s) and render them, create arrays of bullets, hits, and its network variations.

```
const [bullets, setBullets] = useState([]);
const [hits, setHits] = useState([]);

const [networkBullets, setNetworkBullets] = useMultiplayerState("bullets", []);
const [networkHits, setNetworkHits] = useMultiplayerState("hits", []);
```

- useState([]): Initializes local state variables bullets and hits as empty arrays using the useState hook. These states are used to manage the local collection of bullets and hits within the game.
- useMultiplayerState("bullets", []) and useMultiplayerState("hits", []): Utilizes a custom hook (useMultiplayerState) to synchronize state between local and networked multiplayer environments. It initializes networkBullets and networkHits states that will be synced with the multiplayer game state.

```
const onFire = (bullet) => {
  setBullets((bullets) => [...bullets, bullet]);
};

```

- onFire function: When a player fires a bullet, this function is called. It adds a new bullet to the local bullets state array.

```
const onHit = (bulletId, position) => {
  setBullets((bullets) =>
    bullets.filter((bullet) => bullet.id !== bulletId)
  );
  setHits((hits) => [...hits, { id: bulletId, position }]);
};

```

- ❖ onHit function: When a bullet hits a target, this function is called. It updates the local states:
  - ➢ It removes the corresponding bullet from the bullets array.
  - ➢ It adds a new hit (with the id of the bullet and the position of the hit) to the hits array.

```
const onHitEnded = (hitId) => {
  setHits((hits) => hits.filter((h) => h.id !== hitId));
};

```

- ❖ onHitEnded function: When a hit event ends or is resolved, this function is called. It removes the hit with the specified hitId from the hits array.

```
useEffect(() => {
  setNetworkBullets(bullets);
}, [bullets]);

useEffect(() => {
  setNetworkHits(hits);
}, [hits]);

```

- ❖ useEffect hooks: These hooks listen for changes in the local bullets and hits states. When changes occur:
  - ➢ The setNetworkBullets function updates the multiplayer network state with the latest bullets array.
  - ➢ The setNetworkHits function updates the multiplayer network state with the latest hits array.

Code implementation in Experience.jsx

```
const [bullets, setBullets] = useState([]);
const [hits, setHits] = useState([]);

const [networkBullets, setNetworkBullets] = useMultiplayerState(
  "bullets",
  [],
);
const [networkHits, setNetworkHits] = useMultiplayerState("hits", []);

const onFire = (bullet) => {
  setBullets((bullets) => [...bullets, bullet]);
};

const onHit = (bulletId, position) => {
  setBullets((bullets) => bullets.filter((bullet) => bullet.id !== bulletId));
  setHits((hits) => [...hits, { id: bulletId, position }]);
};

const onHitEnded = (hitId) => {
  setHits((hits) => hits.filter((h) => h.id !== hitId));
};

useEffect(() => {
  setNetworkBullets(bullets);
}, [bullets]);

useEffect(() => {
  setNetworkHits(hits);
}, [hits]);
```

3.1.3. 3D Models

3.1.3.1. Character

3.1.3.1.1. Astronaut Bee

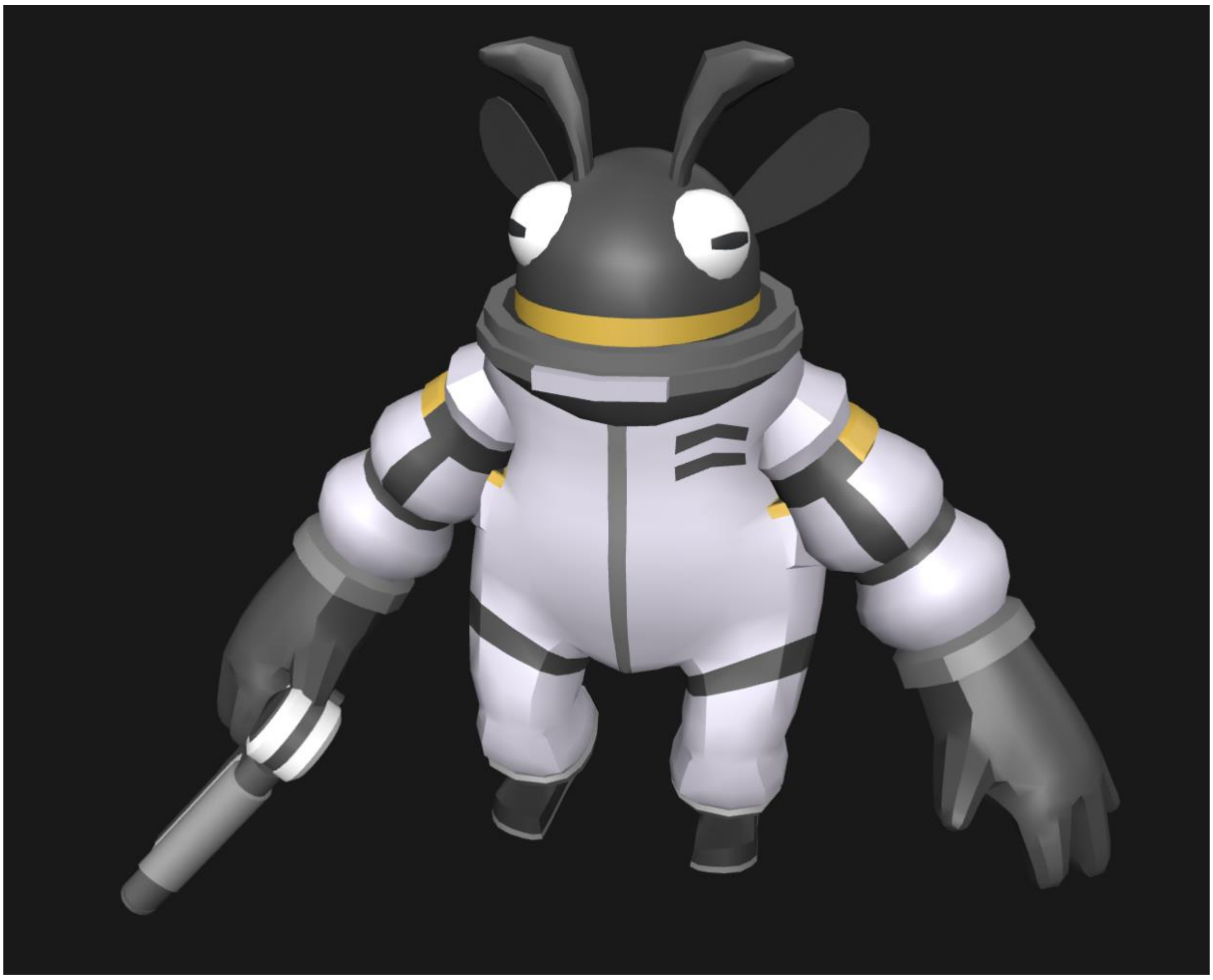

Figure 2. Astronaut Bee Model

3.1.3.1.2. Astronaut Panda

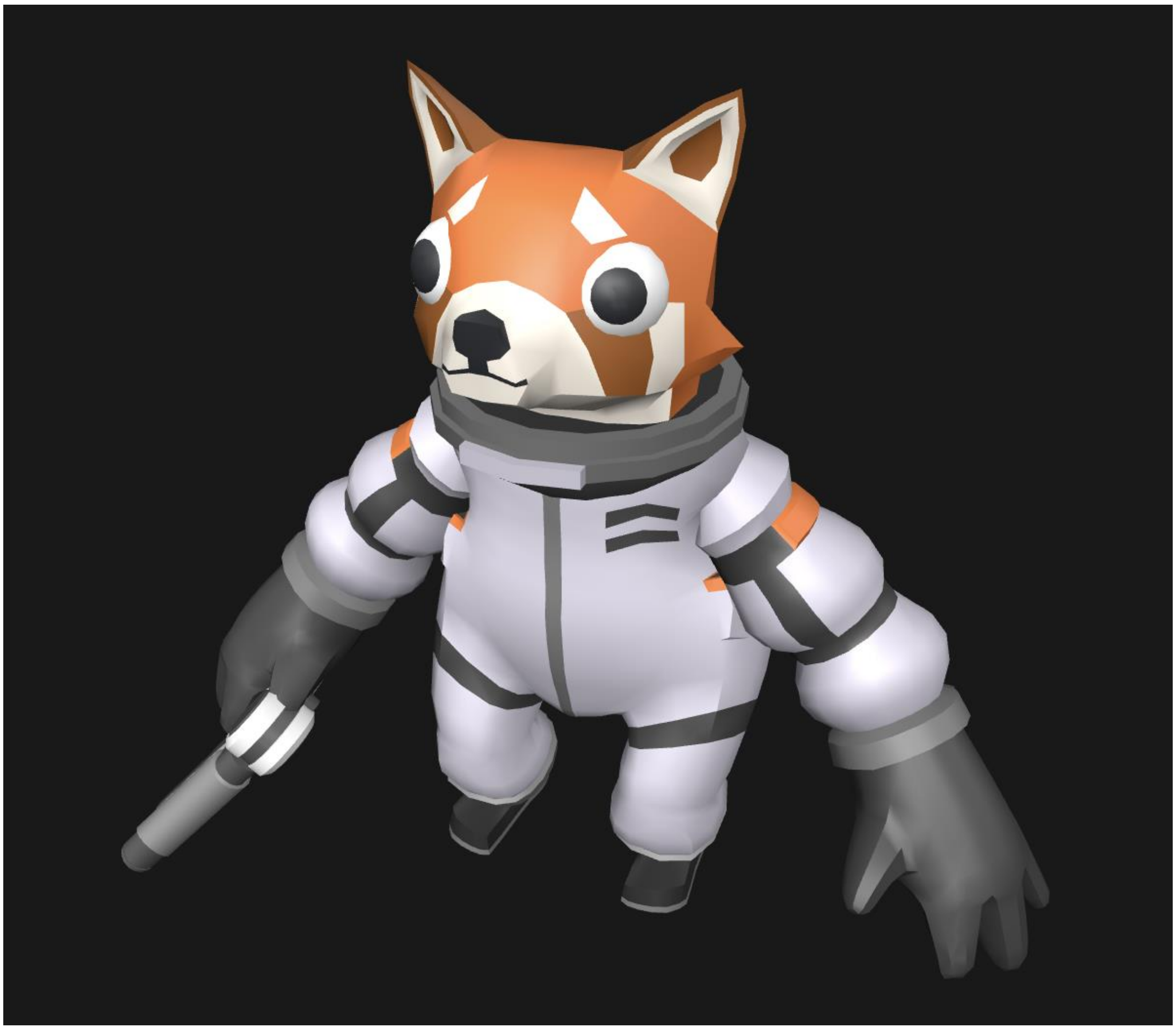

Figure 3. Astronaut Panda Model

<u>3.1.3.1.3. Astronaut Frog</u>

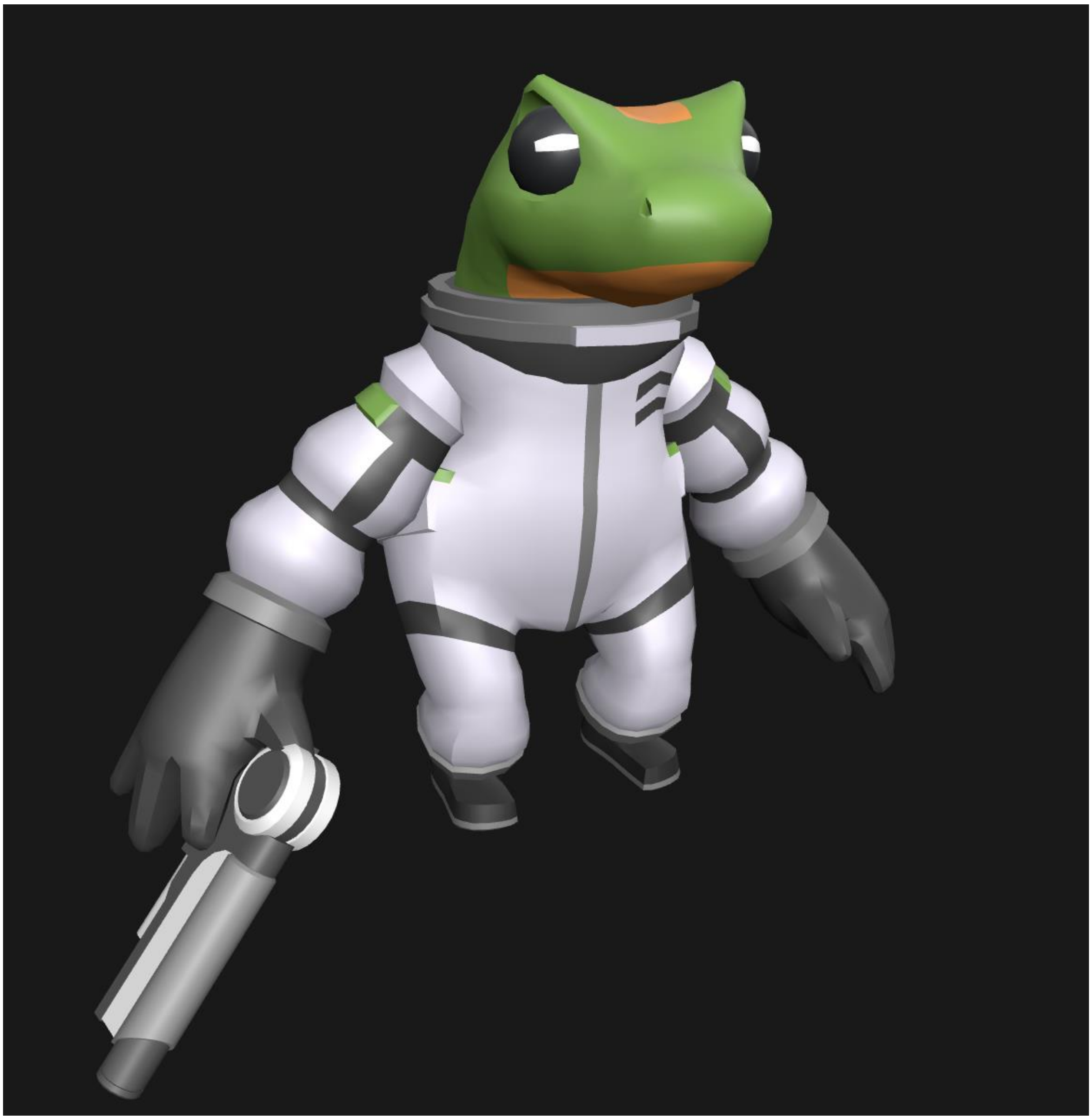

Figure 4. Astronaut Frog Model

3.1.3.2. Bullet

In order to create a bullet, I use the data of player who gets hit(s), the angle and position of the shooter, and the function onHit(), all imported from Experience.jsx, which is the renderer of the game.

```
Bullet = ({ player, angle, position, onHit })
```

Then I define an useEffect hook in React to play firing effect and calculate the bullet's velocity.

```
useEffect(() => {
    // Create and play a firing sound effect
    const audio = new Audio("/audios/rifle.mp3");
    audio.play();

    // Calculate bullet velocity based on angle and set the linear velocity of the rigidbody
    const velocity = {
      x: Math.sin(angle) * BULLET_SPEED,
      y: 0,
      z: Math.cos(angle) * BULLET_SPEED,
    };
    rigidbody.current.setLinvel(velocity, true);
  }, []);

```

- ❖ Inside the useEffect hook:
  - ➢ It creates and plays an audio firing sound effect.
  - ➢ Calculates the bullet's velocity based on the given angle and sets the linear velocity of the rigidbody using setLinvel from the RigidBody component.

In order to render the bullet, I define the bullet as a rigid body (allow game objects to act under physics laws).

```
  <RigidBody
          ref={rigidbody}
          gravityScale={0}
          onIntersectionEnter={(e) => {
            // When the bullet intersects with another object
            if (isHost() && e.other.rigidBody.userData?.type!=="bullet") {
              rigidbody.current.setEnabled(false);
              onHit(vec3(rigidbody.current.translation()));
            }
          }}
          sensor
          userData={{
            type: "bullet",
            player,
            damage: 10,
          }}
        >
          {/* Define the visual appearance of the bullet */}
          <mesh position-z={0.25} material={bulletMaterial} castShadow>
            <boxGeometry args={[0.05, 0.05, 0.5]} />
          </mesh>
        </RigidBody>

```

- onIntersectionEnter event is triggered when the bullet intersects with another object. If it's the host and the intersected object is not a bullet, it disables the rigidbody and triggers the onHit function.

- The mesh represents the visual appearance of the bullet using a box geometry with the defined bulletMaterial.

Finally, I wrap it in a <group/> component, in order to make it based on the position and angle of the shooter.

```
<group position={[position.x, position.y, position.z]} rotation-y={angle}>
</group>
```

3.1.3.3. Map

Firstly, to create the map I personally use Blender for edit GLB models.

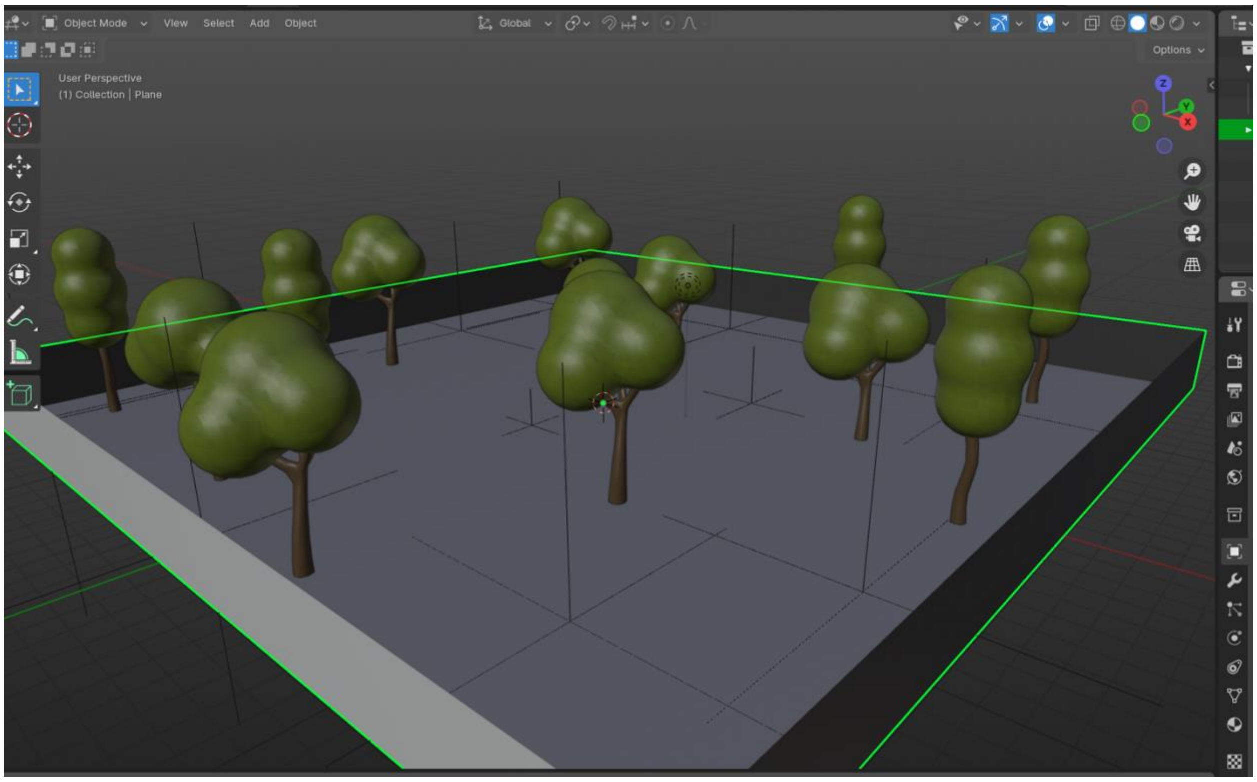

Figure 5. Viewing game map in Blender

Then, in order to render the map, I created the Map(.jsx) component.

```
import { useGLTF } from "@react-three/drei";
import { RigidBody } from "@react-three/rapier";
import { useEffect } from "react";

export const Map = () => {
  const map = useGLTF("models/map2.glb");
  useEffect(() => {
    map.scene.traverse((child) => {
      if (child.isMesh) {
        child.castShadow = true;
        child.receiveShadow = true;
      }
    });
  });
  return (
    <>
      <RigidBody colliders="trimesh" type="fixed">
        <primitive object={map.scene} />
      </RigidBody>
    </>
  );
};
```

```
useGLTF.preload("models/map2.glb");
```

Code Explanation:

- ❖ useGLTF("models/map2.glb") loads a 3D model (in glTF format) named "map2.glb" and stores it in the map variable using the useGLTF hook.

- ❖ Inside the useEffect hook, the code iterates through all the children of the loaded 3D model's scene using map.scene.traverse(). For each child object in the scene, if it is a mesh (child.isMesh), it sets properties to enable casting and receiving shadows (child.castShadow = true; child.receiveShadow = true;).

- ❖ RigidBody component is used, which presumably adds physics properties to the 3D object (the map) represented by map.scene.

- ❖ Inside the RigidBody component, a colliders="trimesh" attribute seems to indicate the type of collider used for physics interactions, and type="fixed" might specify that this rigid body is immovable.

- ❖ Finally, useGLTF.preload("models/map2.glb"); preloads the specified 3D model to ensure it's ready to be used without delay when needed.

3.1.4. Leaderboard

Firstly, I import userPlayerList() from the Playroom Kit library to fetch the list of current player(s) then I map it to a “players” variable.

```
import { usePlayersList } from "playroomkit";
const players = usePlayersList(true);
```

Then, with every existing player, I generate a div. With each div, it shows name, kill(s), death(s). Also, a top-right button to toggle fullscreen using native JavaScript code.

```
<div className="fixed top-0 left-0 right-0 p-4 flex z-10 gap-4">
  {players.map((player) => (
    <div
      key={player.id}
      className={`bg-white  bg-opacity-60 backdrop-blur-sm flex items-center rounded-lg gap-2 p-2 min-w-[140px]`}
    >
      <img
        src={player.state.profile?.photo || ""}
        className="w-10 h-10 border-2 rounded-full"
        style={{
          borderColor: player.state.profile?.color,
        }}
      />
      <div className="flex-grow">
        <h2 className={`font-bold text-sm`}>
          {player.state.profile?.name}
        </h2>
        <div className="flex text-sm items-center gap-4">
          <p>⍰üî´ {player.state.kills}</p>
          <p>⍰üíÄ {player.state.deaths}</p>
        </div>
      </div>
    </div>
  ))}
</div>
<button
  className="fixed top-4 right-4 z-10 text-white"
  onClick={() => {
    // toggle fullscreen
    if (document.fullscreenElement) {
      document.exitFullscreen();
    } else {
      document.documentElement.requestFullscreen();
    }
  }}
>

```

3.2. Working Environment

- IDE:
    - Iterm2 (Terminal Emulator)
    - Neovim
- Dependencies:
    - npm
    - yarn
- Programming language(s): JavaScript
- Hardware: Macbook Pro (2020)
- Operating System: Unix

CHAPTER 4 : RESULTS

4.1. Create player(s)

If a user successfully follows the game link (or runs locally on computer), it should show the game lobby.

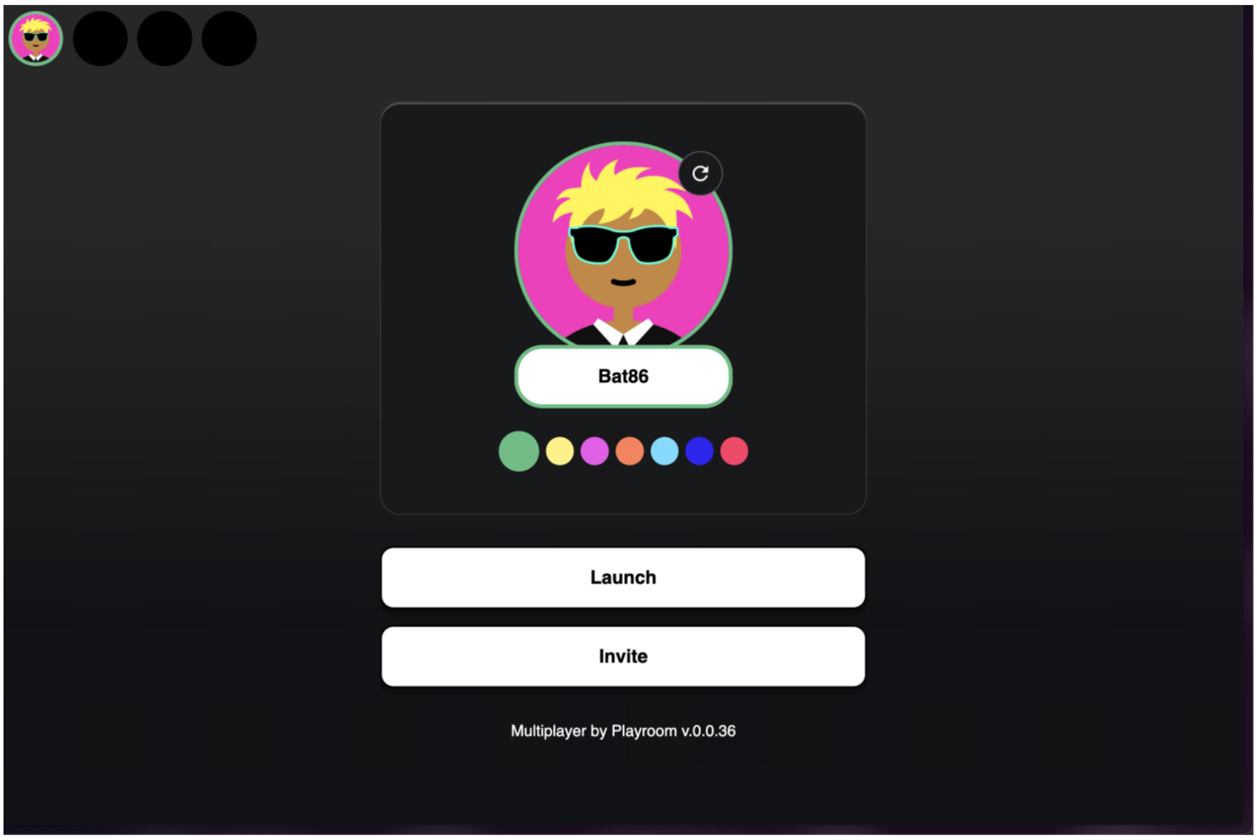

Figure 6. Game Lobby

If the user decides to host a game with other people, he/she could press on Invite button to create a link and a QR code.

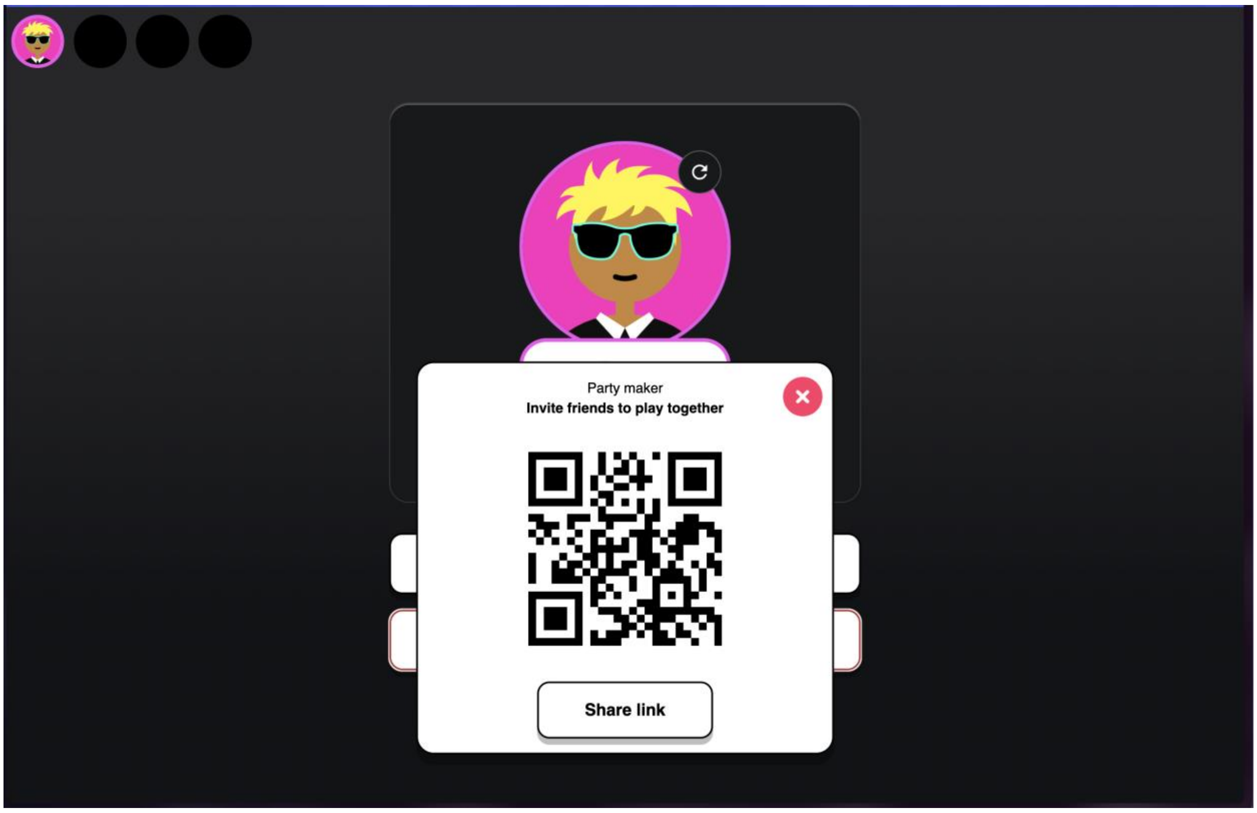

Figure 7. Game Lobby Invite

When user click on Launch button, it should renders the scene, the map and the player's character.

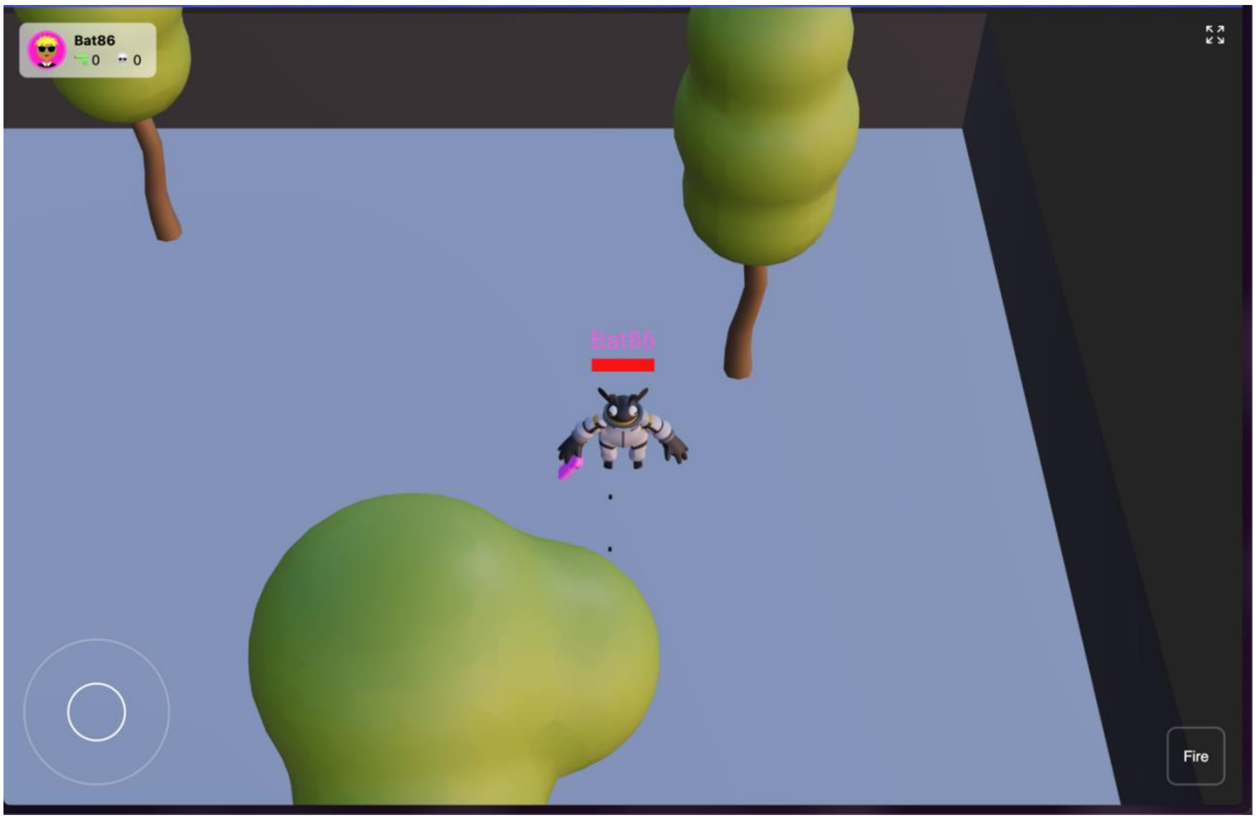

Figure 8. Game Screen

4.2. Firing bullet(s)

When the user launches the game correctly and he/she is viewing the game screen, press on the Fire button should see the bullet flying.

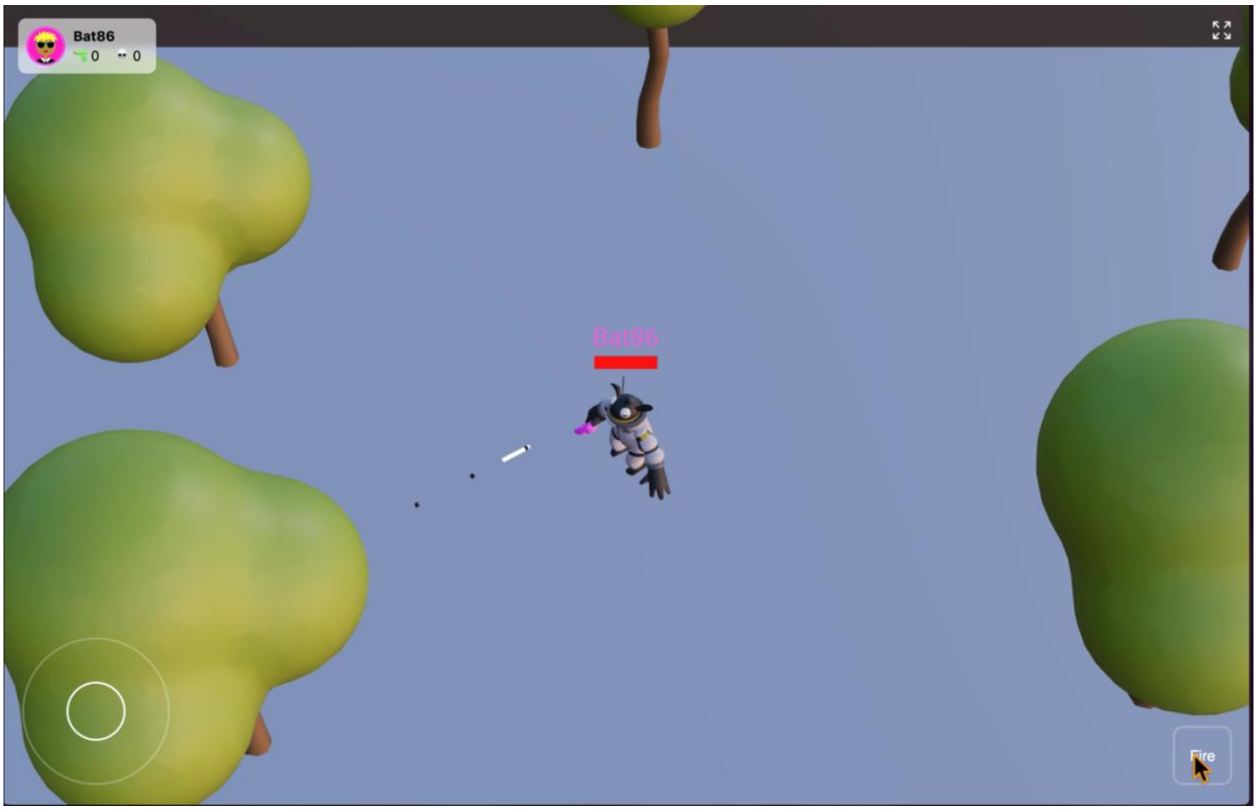

Figure 9. Bullet View

If the user shoots at other player, the bullet should hit the other player and display its effect.

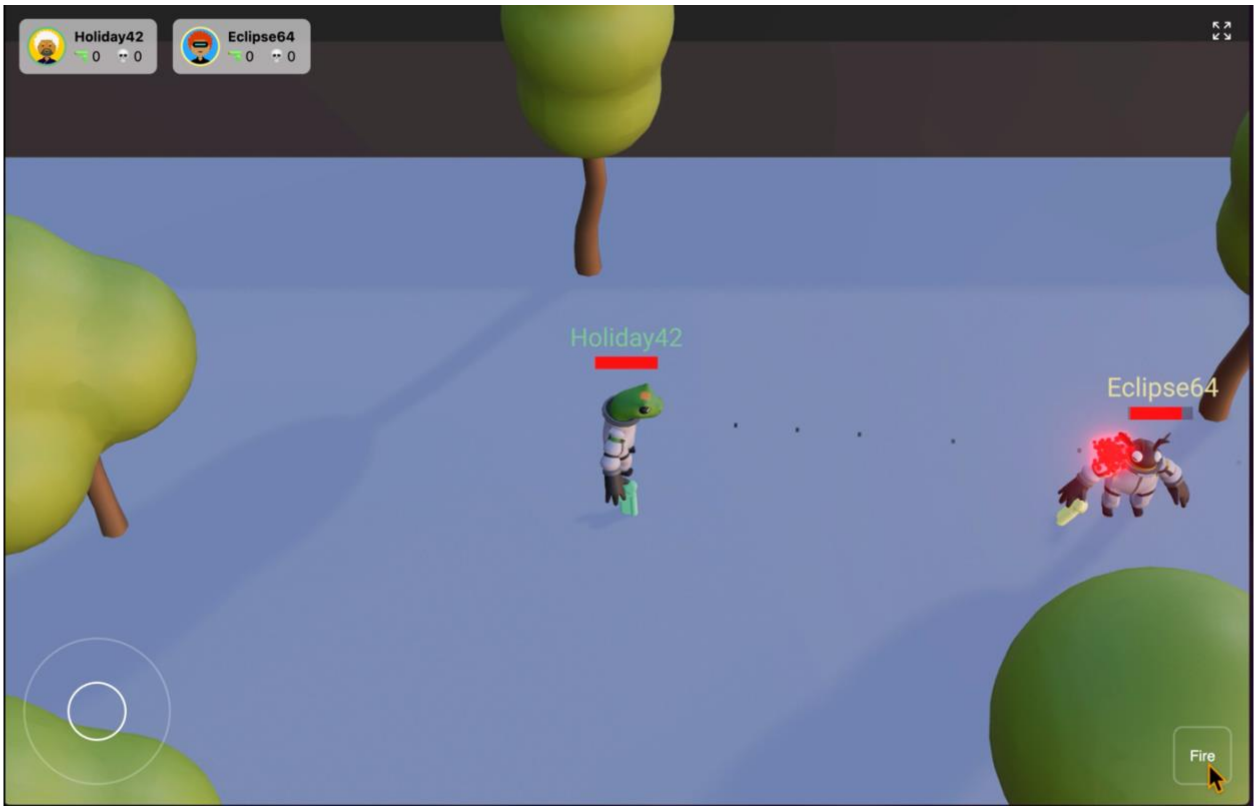

Figure 10. Bullet Hits View

4.3. Leaderboard

When a user(s) enters the game, the leaderboard should be displayed on top left of the viewport.

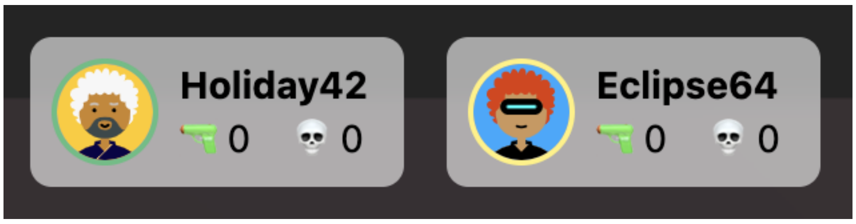

Figure 11. Leaderboard

When a user killed a player, he/she gains 1 point for killing. Likewise, the defeated player gains 1 point for dying.

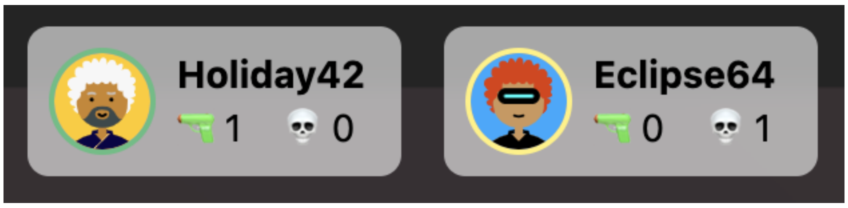

Figure 12. Leaderboard: Player Point(s) Example

4.4. Character Animations

4.4.1. Idle

When the joystick is not pressed, the character's animation is idle.

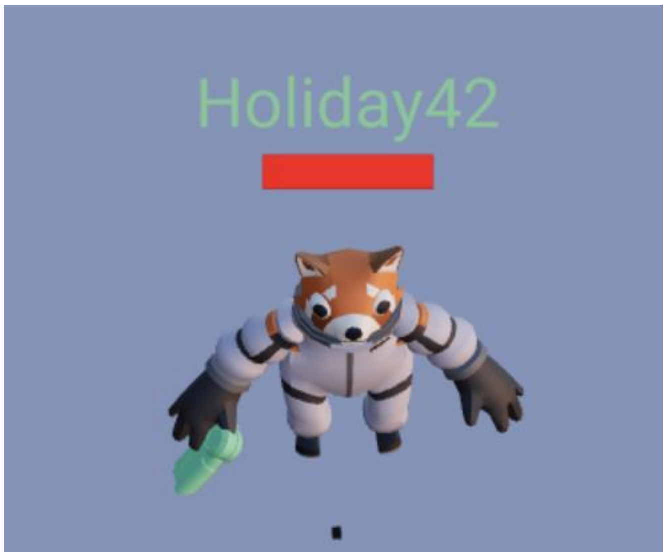

Figure 13. Character Animation: Idle

4.4.2. Running

When the joystick is pressed and dragging in any direction, the character's animation is running.

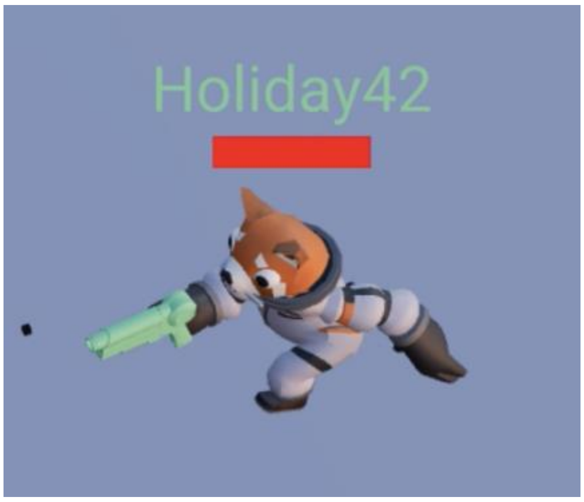

Figure 14. Character Animation: Running

4.4.3. Death

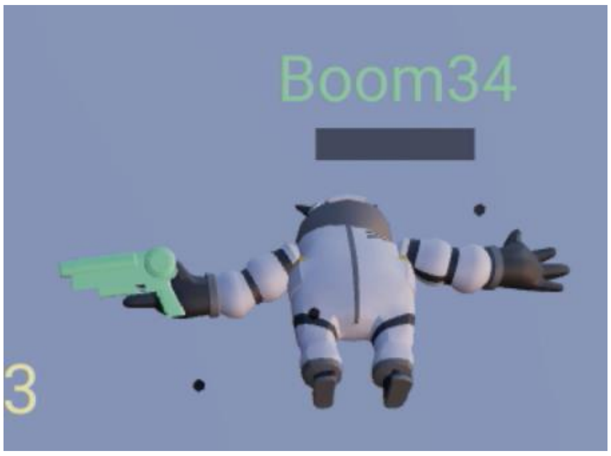

Figure 15. Character Animation: Death

4.4.4. Idle & Shooting

When the joystick is not pressed and the Fire button is pressing, the character's animation is idle and shooting combined.

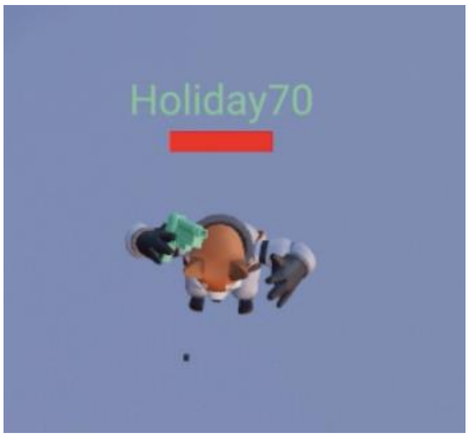

Figure 16. Character Animation: Idle & Shooting

4.4.5. Running & Shooting

When both the joystick is dragging and the Fire button is pressing, the character's animation is running and shooting combined.

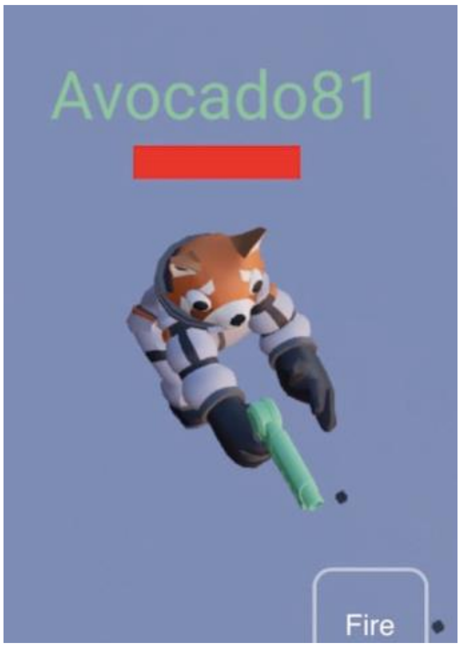

Figure 17. Character Animation: Running & Shooting

CHAPTER 5 : CONCLUSION

5.1 Conclusion

On the whole, I take immense pride and deep satisfaction in our project. Reflecting on what has been accomplished, the practical exposure to cutting-edge technology, the hurdles conquered, regardless of internal team issues or the eventual assessment of my 3D multiplayer game, I will always value this endeavor. Be it the newfound knowledge of utilizing innovative tools, the collaborative experience, or the trials encountered in implementing various features, I believe that everything learned and faced during this project will remain a valuable part of my future endeavors in the field.

5.2 Future work

Upcoming developments will encompass, but won't be restricted to:

- AI-driven bots.
- Allowing users to import map and character models.
- Expanding the map limitlessly.
- Introducing additional game modes like Survival Mode.

These future enhancements aim to diversify the website's offerings, prioritizing the inclusion of these features while ensuring the website's stability. Our ultimate objective for this platform is to make it appealing to both small businesses and individuals seeking short-term contract opportunities.

REFERENCES

1. Neovim - Neovim
2. GitHub - GitHub
3. Fork - Fork
4. ReactJS - ReactJS
5. Three.js - Three.js
6. React Three Fiber - React Three Fiber
7. React Three Drei - React Three Drei
8. React Three Rapier - React Three Rapier
9. Playroom Kit - Playroom Kit
10. Blender - Blender
11. gltfjsx - gltfjsx