VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

FINAL REPORT

Team Project: Bug Brawl

By Group 10 – Member List

No.	Full Name	Student ID	Contribution
1	Nguyễn Thị Chung Anh	ITITIU21155	50%
2	Hà Đỗ Tây Đô	ITITIU21177	50%

Course: Net-Centric Programming
Instructor: MSc. Lê Thanh Sơn

Ho Chi Minh City, Vietnam
2025

Table of Contents

ABSTRACT

Bug Brawl is a simple multiplayer web game that allows up to four players to join a match and play together in real time. Players can create or join game rooms using a room code or find opponents through a matchmaking system. The game is built using web technologies like React.js for the frontend and WebSockets for real-time communication between players.

The project focuses on basic multiplayer features such as room creation, player management, and live game state updates. Players can input a name, join a room, and wait for others before the host starts the game. The interface is designed to be clean and user-friendly, with basic animations and layout components.

This project demonstrates the core concepts of real-time multiplayer functionality in a browser-based environment and serves as a foundation for future development of more complex gameplay mechanics.

LIST OF FIGURES

CHAPTER 1:
INTRODUCTION

1. Overview

Bug Brawl is a multiplayer game in the browser, which is played by players who compete against each other and try to be the last one standing. In this case, the game combines sabotage features with question-based player interaction to create a completely unpredictable and entertaining experience.

2. Objective

The project's primary goal was to make a multiplayer game that would efficiently use WebSocket communication at first. The primary goals were, of course, that clients are in real-time sync, control game stability when there are several players, and make the game enjoyable.

3. Team members and roles

- Nguyễn Thị Chung Anh: Full-stack Developer and UI/UX Design
- Hà Đỗ Tây Đô: Backend Developer

CHAPTER 2:
PREREQUISITES AND SETUP GUIDE

1. Prerequisites

- **Node.js** (version ≥ 18.0.0) – for running the React frontend.
- **Go (Golang)** (version ≥ 1.20) – for running the backend WebSocket server.
- **Git** – for cloning the project repository.
- **Modern web browser** – such as Google Chrome or Firefox for playing the game.

2. Setup guide

2.1 Download or Clone the Project

2.1.1 Clone via GitHub (Recommended)

If you have Git installed, run the following commands in your terminal:

git clone https://github.com/Wtlm/Bug_Brawl.git
cd bug-brawl

2.1.2 Download as Zip (Alternative)

1. Go to the GitHub project page: *https://github.com/Wtlm/Bug_Brawl.git*
2. Click the green Code button.
3. Select Download ZIP.
4. Extract the ZIP file to your desired location.
5. Open the extracted folder.

2.2 Backend Setup

1. Navigate to the server folder: cd server
2. Make sure all Go dependencies are downloaded: *go mod tidy* .
3. Run the server: *go run* .

The backend server runs at: **http://localhost:8080**

2.3 Frontend Setup

1. In the new terminal, navigate to the client folder: *cd client/vite-project*
2. Install dependencies: *npm install*
3. Start the development server: *npm run dev*

The client will be available at: **http://localhost:5173**

2.4 Environment and Port Configuration

This project uses default ports, ensure these ports are not in use before running the project:

- 5173 for the React frontend
- 8080 for the Go backend

2.5 Running the Game

1. Open **http://localhost:5173** in your browser.
2. Enter a nickname and click Create Room or Join Room.
3. Copy the room ID to share with other players.
4. When all players are ready, the host can start the game.
5. Multiple players can connect using different tabs or devices.
6. Players can click Find Match to match with 3 random players

CHAPTER 3:
SYSTEM OVERVIEW

1. Description

Up to four players can enter a room, solve the problems, and apply graphic trappings to defeat other players in the fast-paced, sabotage-inspired game Bug Brawl. If a player's health meter drops to zero, they will be eliminated from the game, and the last stand player will be the winner.

2. Key features

- **Multiplayer gameplay:** Players can join a match with up to four participants.
- **Matchmaking (create/join/find):** Players can create rooms, join via room code, or be matched randomly.
- **Real-time WebSocket communication:** WebSocket is used for bi-directional real-time communication between clients and the server.
- **Sabotage mechanics:** Players can trigger sabotage effects on others after correctly answering questions.
- **Health and elimination system:** Each player has a health bar; wrong or slow answers and sabotage reduces health.

3. Technologies used

- **Health and elimination system:**
  - Frontend: React.js, Tailwind CSS, Vite, CSS
  - Backend: Go, Gorilla WebSocket, Gorilla Mux
  - Tools: Figma (UI design)
- **Repository and Distribution:**
  - Source Code: Available on Github (main branch)

4. Data flow overview

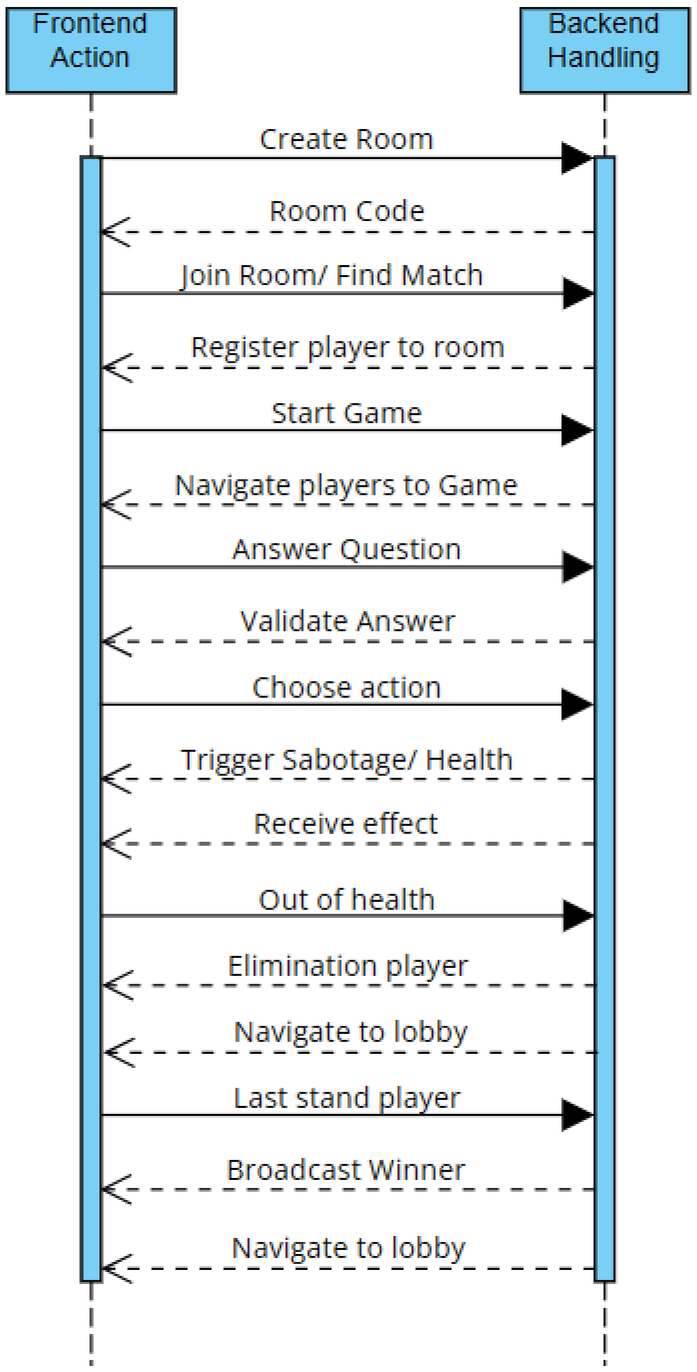


Figure 2-1: Data flow diagram

CHAPTER 3:
RESULT

1. Summary of Implemented Features

- Room creation/joining: In this step, players have the option of either opening a room and assuming the role of the host or joining an already existing one by entering the room code. The leader can then initiate the game as soon as everyone is prepared."
- Real-time player updates: Player status, health, and actions are shared with all clients by WebSocket in real time. Thus, there is no stop-and-go multiplayer communication.
- Game state synchronization
- Basic sabotage and question system: Players who properly answer the questions gain the ability to sabotage the enemies by making them blurred, flashed, etc. The changes are real-time.
- Health-based elimination and win conditions: At the start of the game, every player gets full health. Transfer of energy and wrong answers result in the loss of health. This, in turn, causes elimination. The person who remains at the end is seen as the winner.
- Endgame announcement: The end of a game occurs when there is only one player left, where everything stops and a message is broadcast across all displays, which introduces the winner and then brings everyone back to the home screen.

2. Final Screenshot

Upon launching the web, the Opening Screen (see *Figure 6-1*) will appear, displaying the app's logo for a few seconds. During this time, the system is initializing essential data in the background.

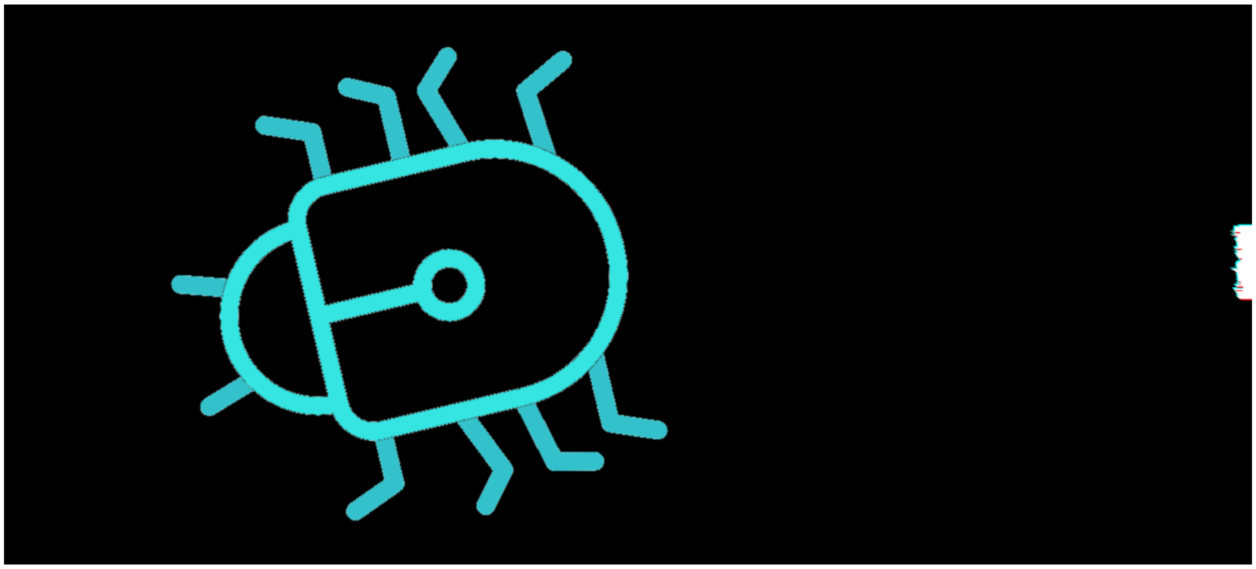


Figure 5-1: Opening Screen

Following the Opening screen, players are directed to the Start Game screen(*Figure 5-2*).

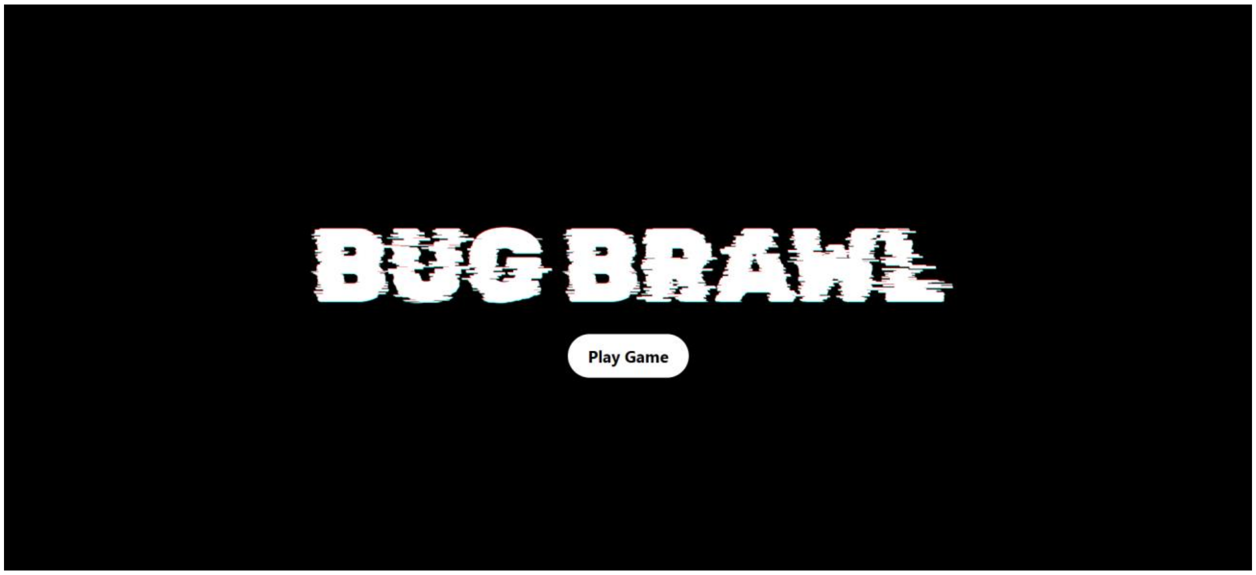


Figure 5-2: Start Game Screen

Click Play Game to connect Web Socket and navigate to Lobby Screen (Figure 5-3).

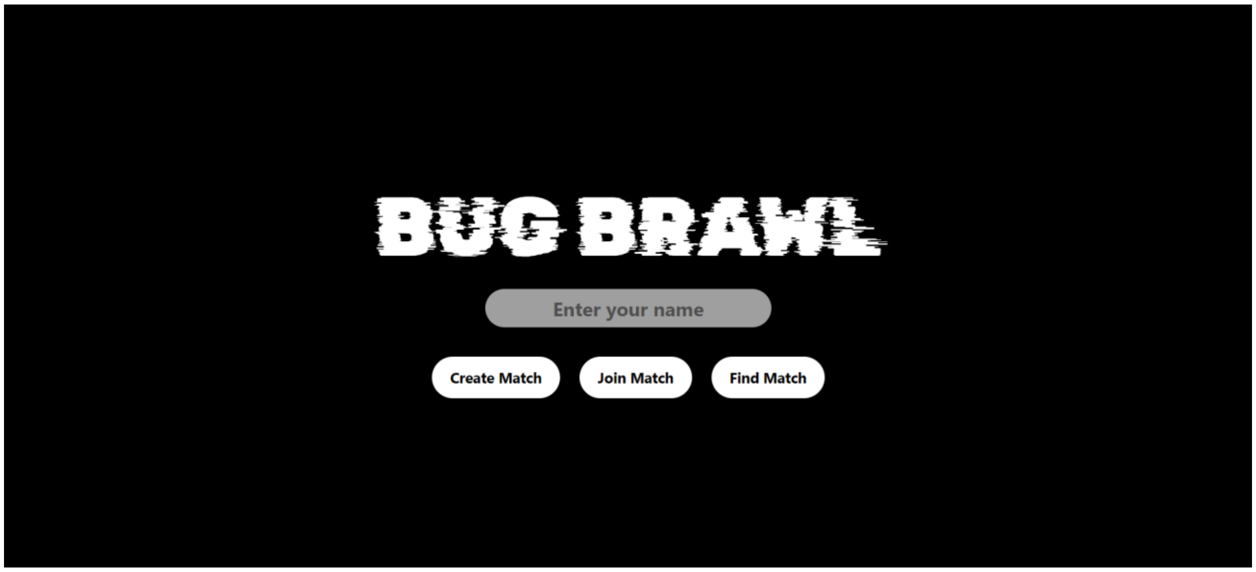


Figure 5-3: Lobby Screen

Click the Create Match button to send a new room request to the server, and then the server will respond to the player with a room code (Figure 5-4).

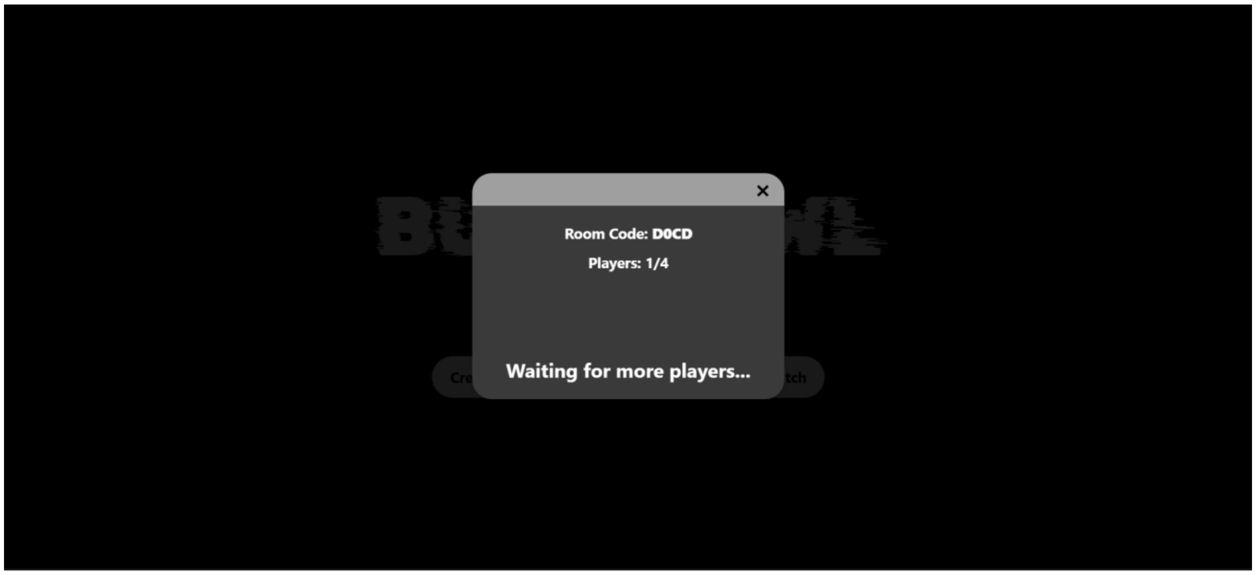


Figure 5-4: Create Room Screen

When other players join the room, the server will update the player count of their room. If more than one player is in the room, the host can click Start Game to send a request to start the game to the server (Figure 5-5).

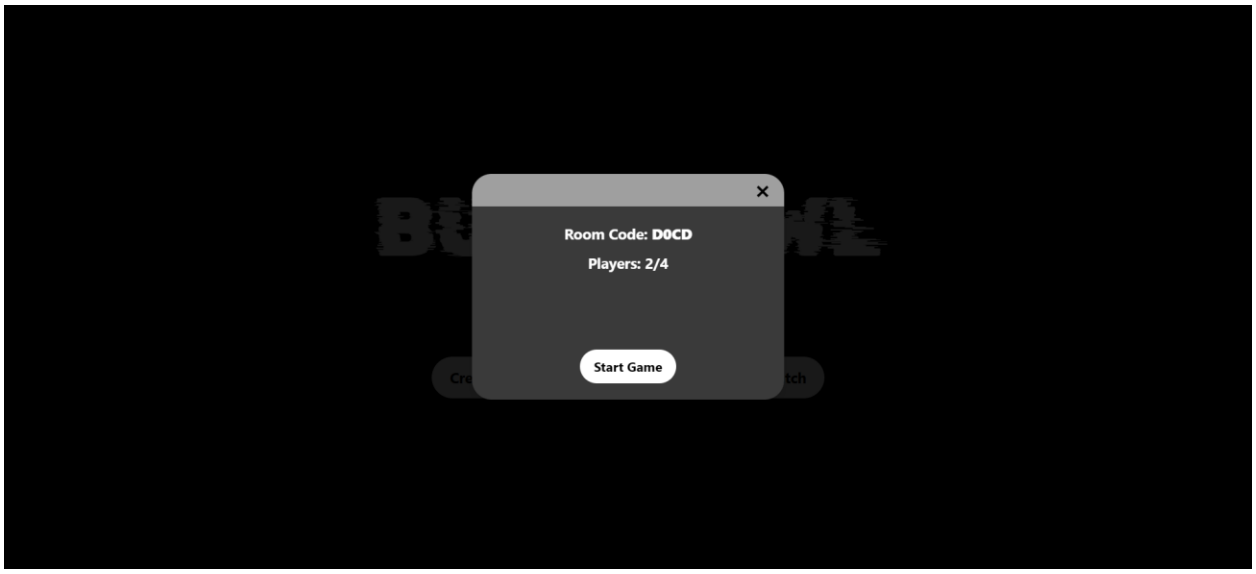


Figure 5-5: Host Screen to start the game

To join the room with friends, other players will click on the Join Match button and enter the room code to send a joining room request, and then the server will append players to the room and respond with the current number of players in the room (Figure 5-6).

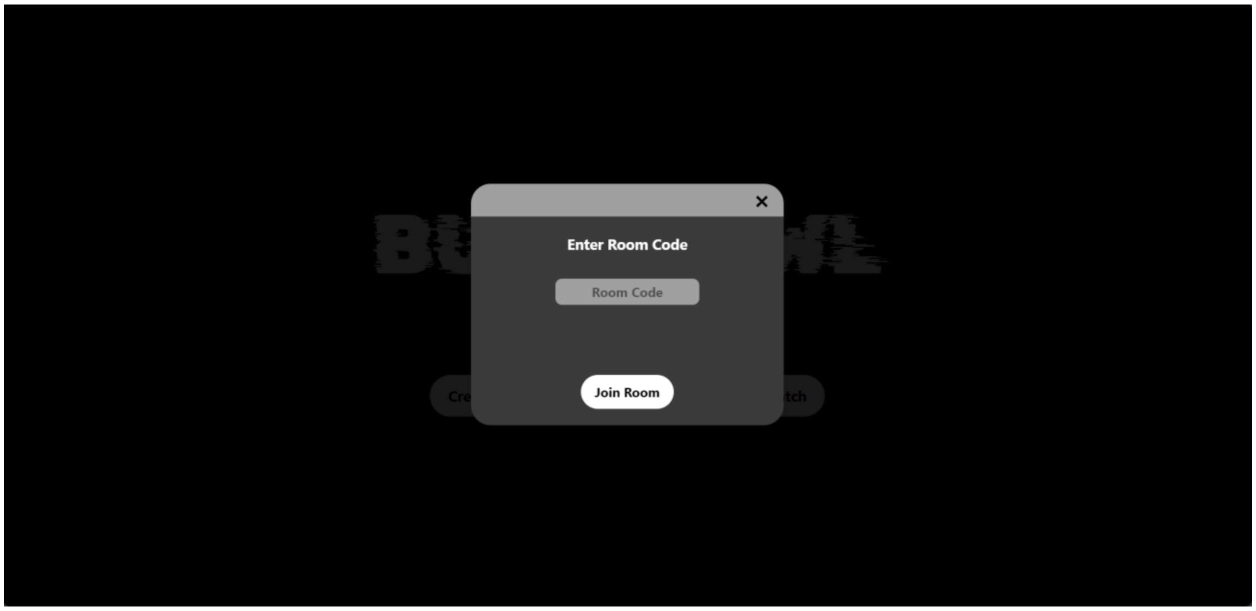


Figure 5-6: Join Room Screen

The server will respond to the room with the room code to the unhost player. Players can leave the room by clicking the Leave Room button, and the server will remove the player from the room and update the player count of the room (Figure 5-7).

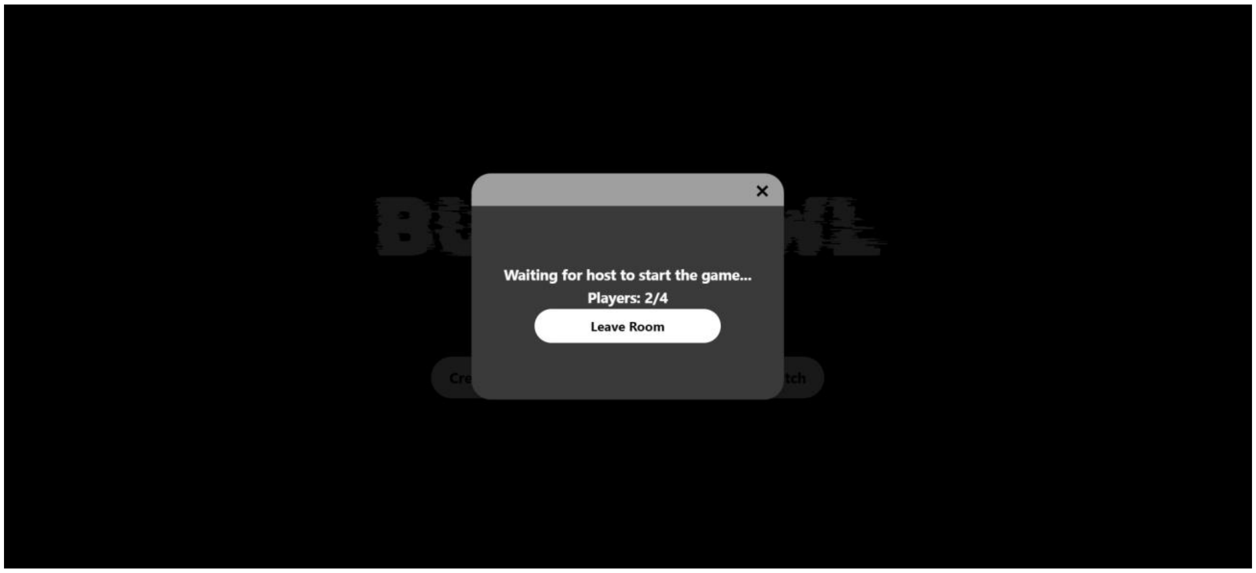


Figure 5-7: Unhost Screen in Room

The player can also click the Find Match button to join the match with 3 three random players (Figure 5-8).

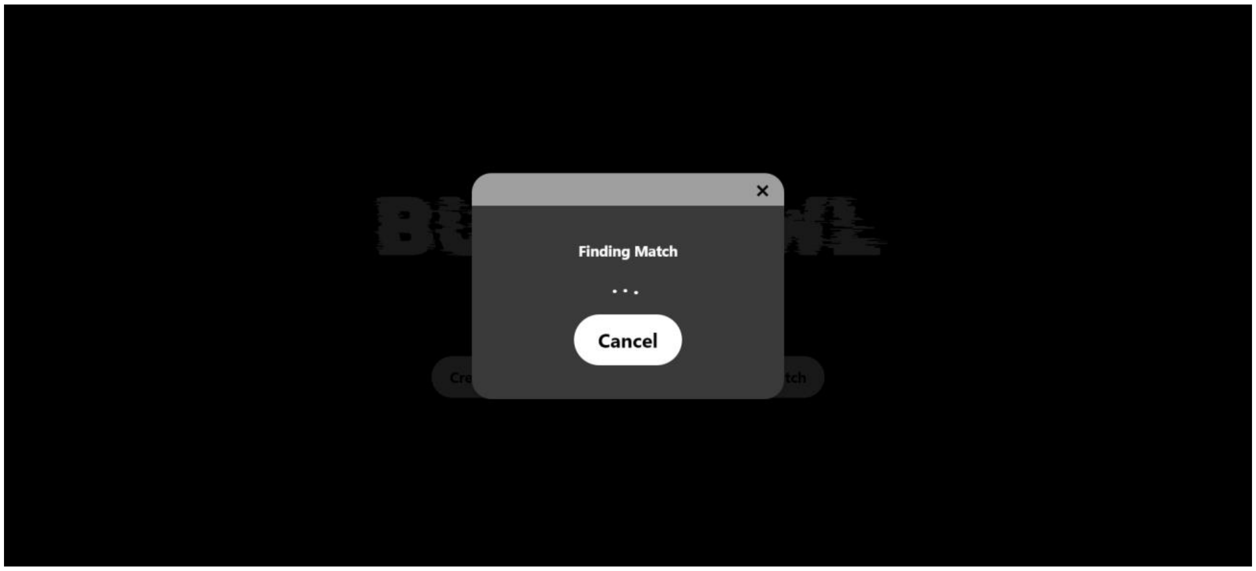


Figure 5-8: Find Match Screen

When there are enough players, the server will send a countdown response (Figure 5-9) and navigate players to the Game Screen (Figure 5-10).

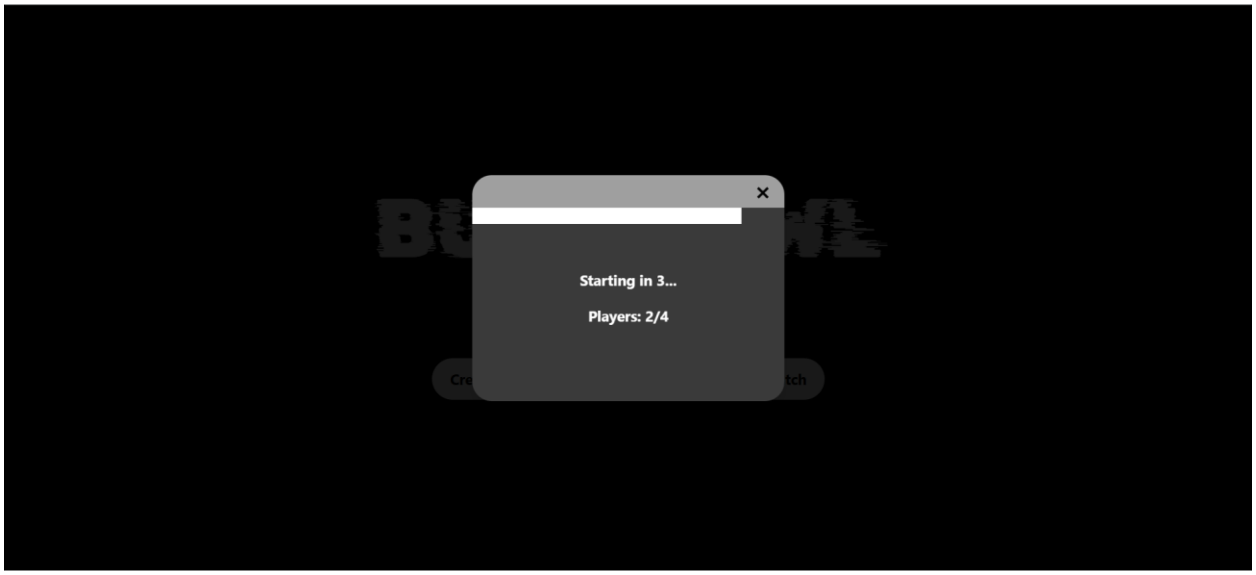


Figure 5-9: Match Found Screen

The game screen shows player’s health state.

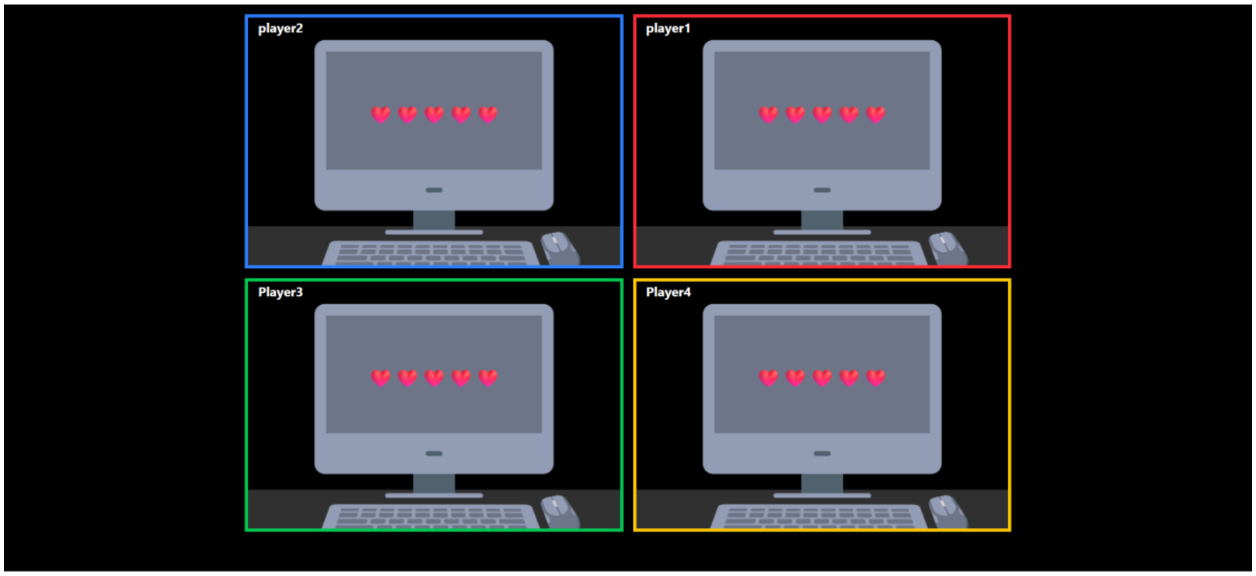
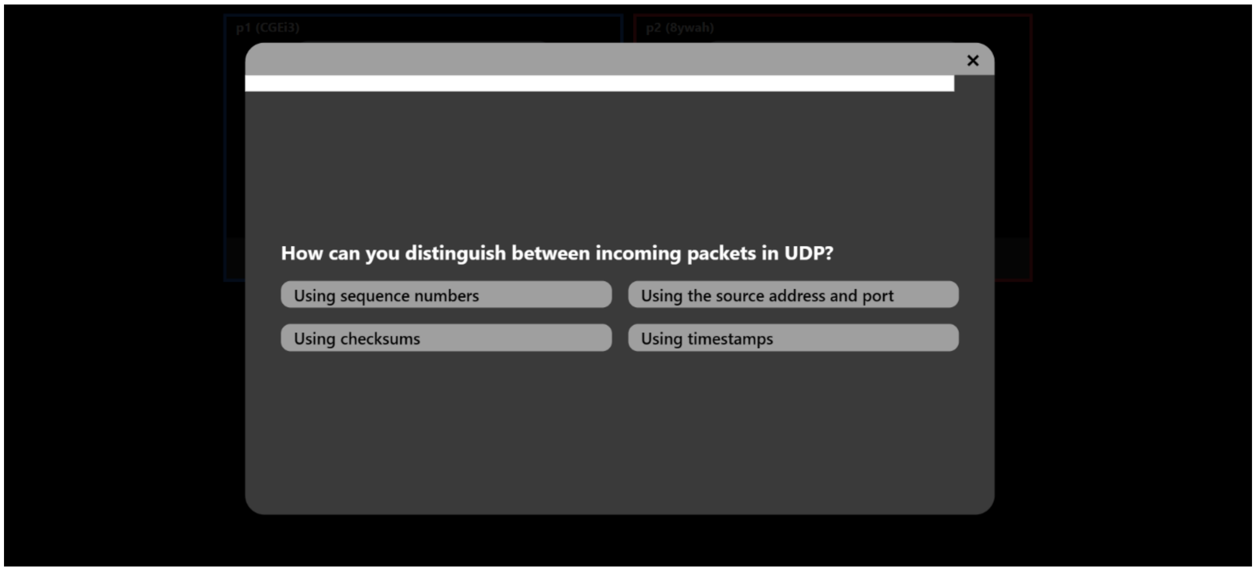


Figure 5-10: Game Screen

When the game starts, there will be a pop-up to display multiple-choice question (Figure 5-11)


How can you distinguish between incoming packets in UDP?

Using sequence numbers

Using the source address and port

Using checksums

Using timestamps

Figure 5-11: Question Popup

When there is one player who chooses the correct answer fastest, a notification pop-up appears for every player (Figure 5-12).

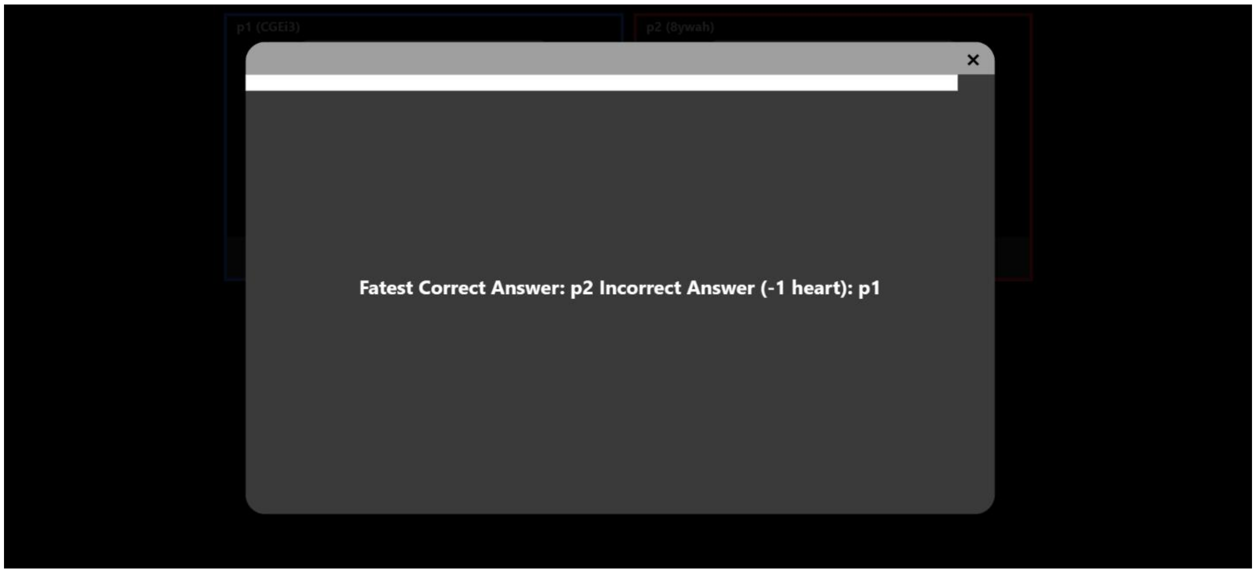

Fatest Correct Answer: p2 Incorrect Answer (-1 heart): p1

Figure 5-12: Fastest Correct Answer Popup

The fastest player will have a list of sabotage to choose to apply for other players (Figure 5-13).

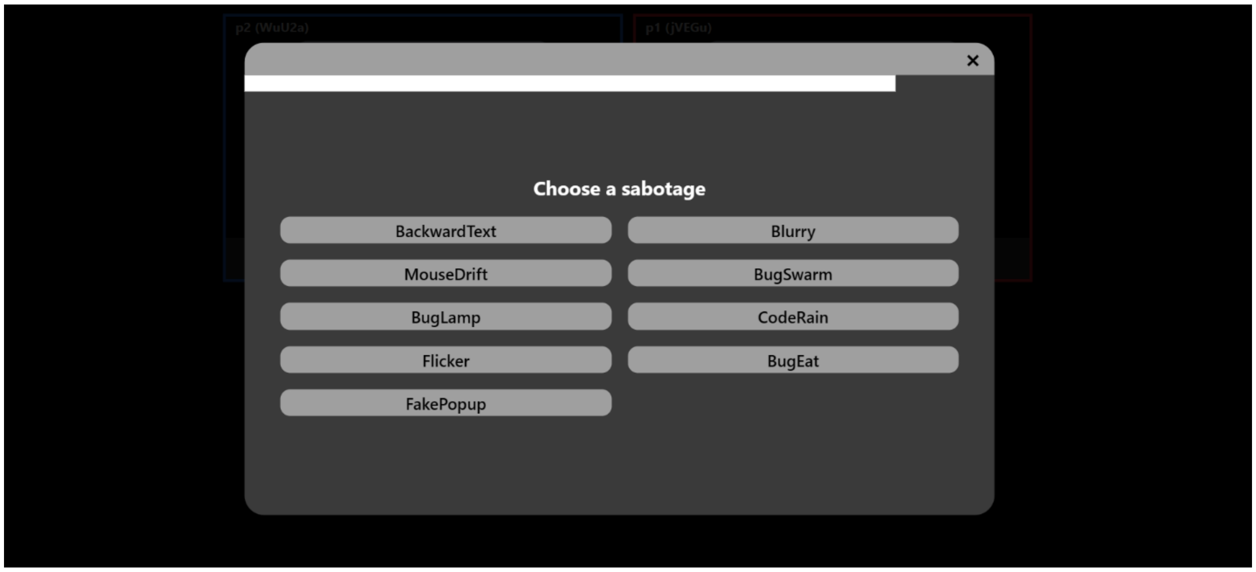


Figure 5-13: Choose Sabotage Popup

Other players need to wait until the player who has the fastest correct answer to choose sabotage (Figure 5-14).

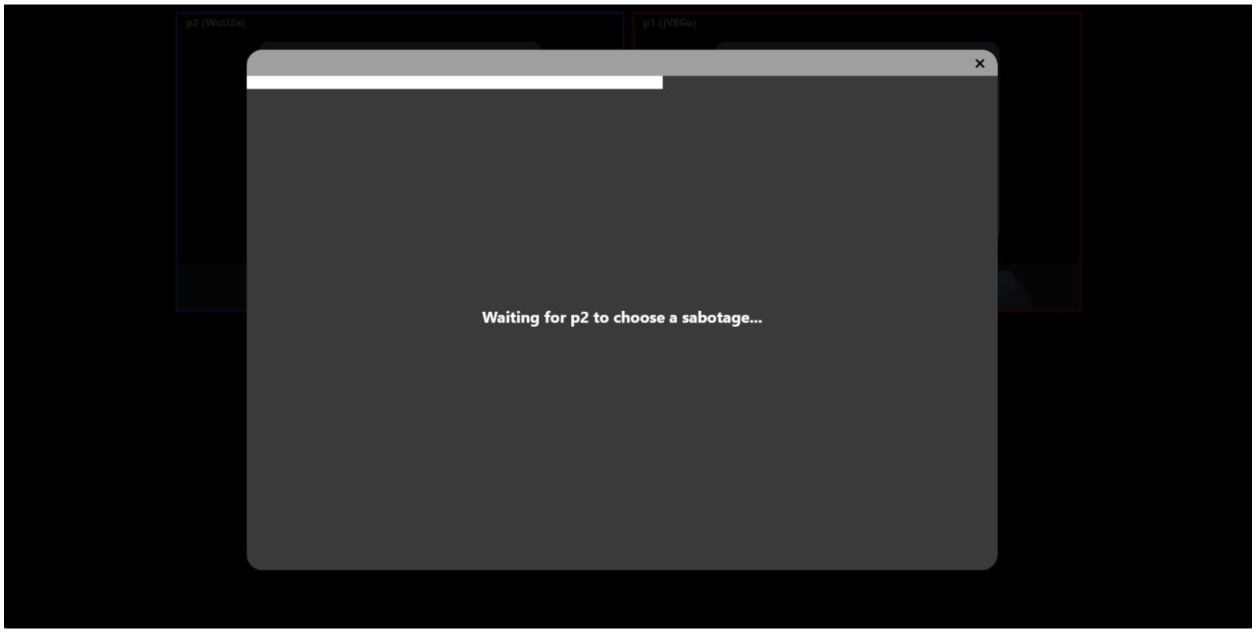

Waiting for p2 to choose a sabotage...

Figure 5-14: Waiting for the player to choose sabotage

In case no one in the room has the correct answer (Figure 5-15), every player will be assigned one random effect.

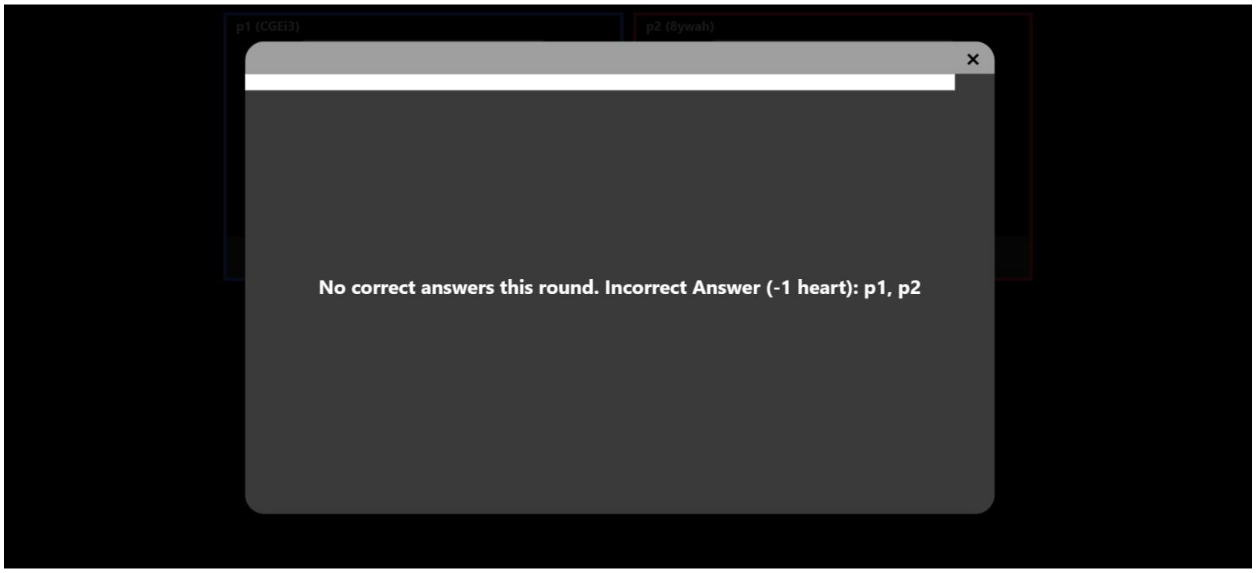
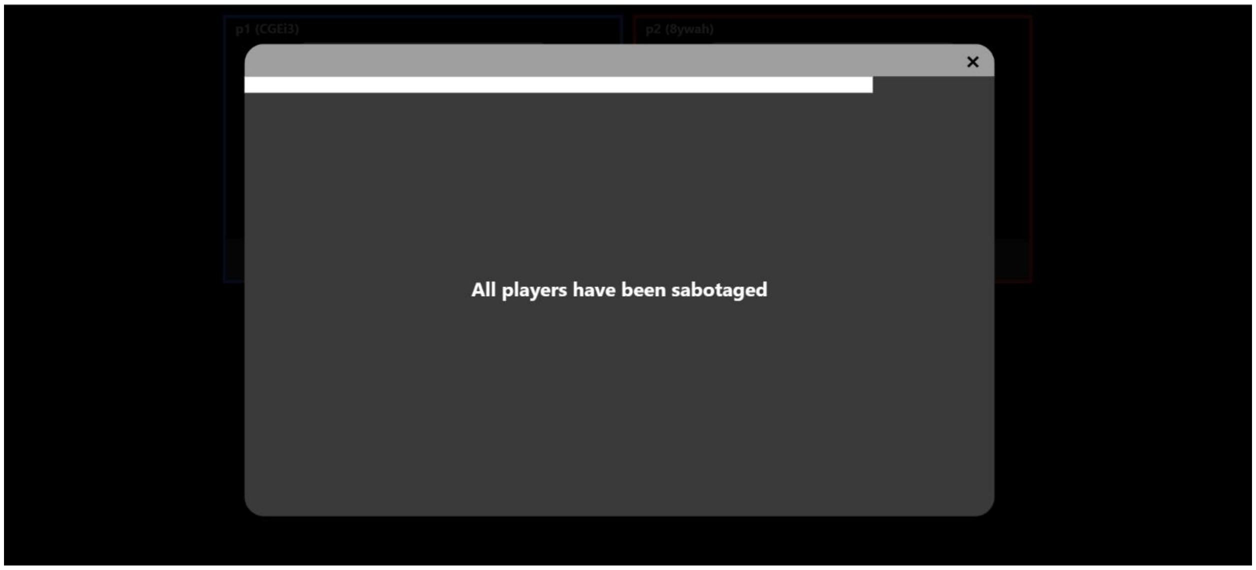

No correct answers this round. Incorrect Answer (-1 heart): p1, p2

Figure 5-15: No Correct Answer Popup

The system will notify all players that they have been sabotaged (Figure 5-16)..



All players have been sabotaged

Figure 5-16: Sabotage applied to all players Popup

Below are samples of all the effects that the game has (Figure 5-17 to 5-24).

There are a swarm of bugs that start appearing on the sabotaged player's screen. They crawl randomly across the screen, increasing in number over time.

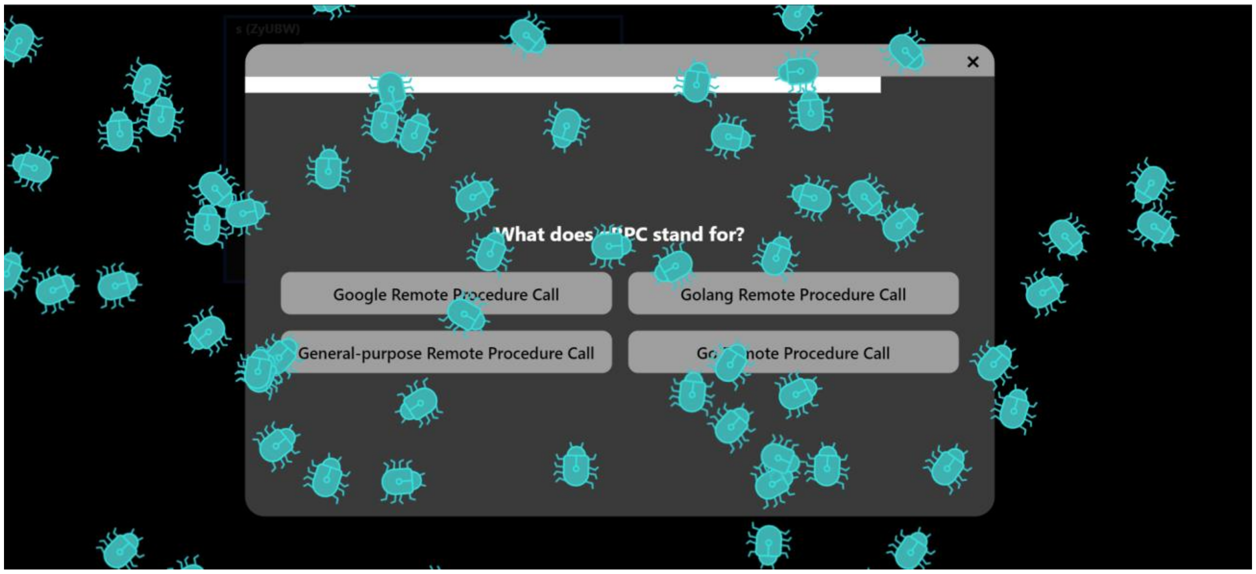
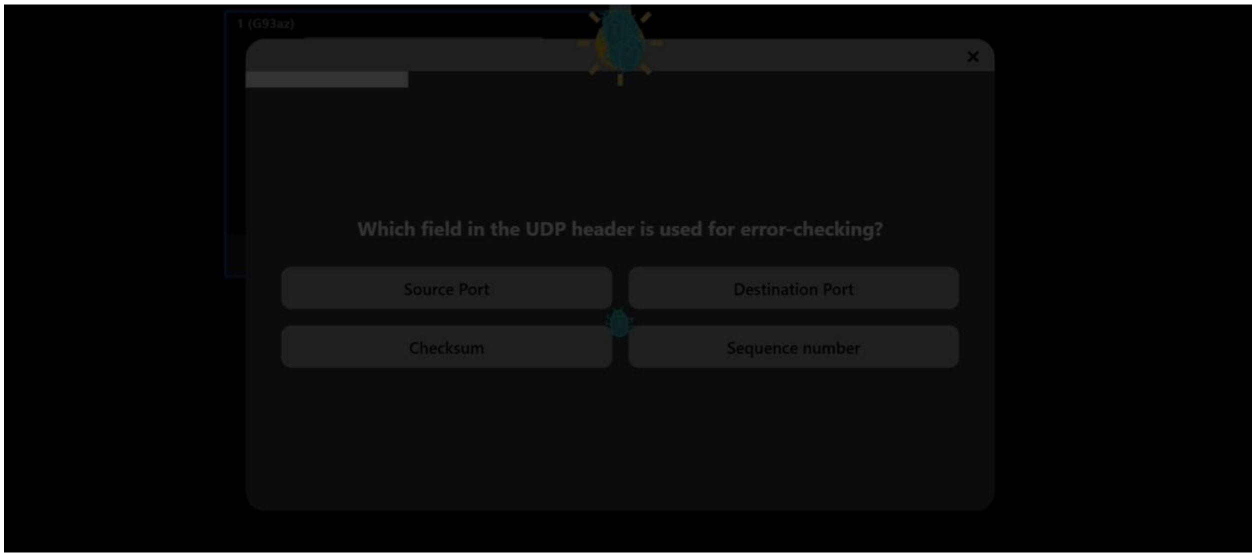

s (ZyUBW)
×
Google Remote Procedure Call
Golang Remote Procedure Call
General-purpose Remote Procedure Call
Go note Procedure Call
What does PC stand for?

Figure 5-17: Bug Swarm Effect

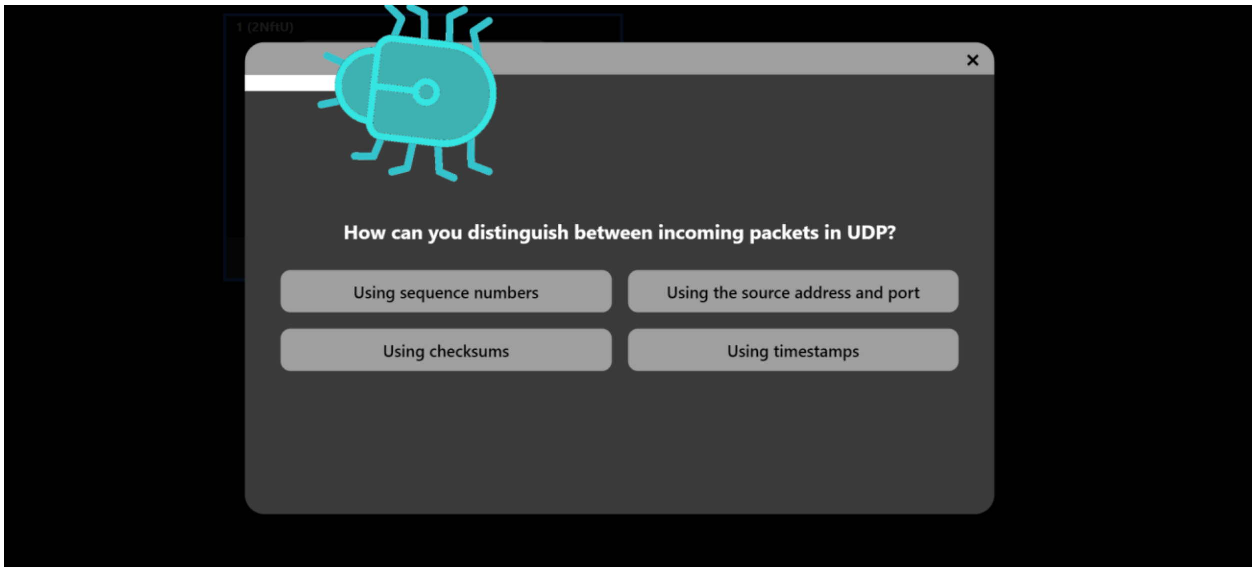
Bugs start flying in from off-screen toward the bulb on the top of the screen. As more bugs gather around the bulb, the screen slowly dims, simulating the bugs blocking the light source.



Which field in the UDP header is used for error-checking?

Figure 5-18: Bug Lamp Effect

A giant bug crawls into view from the side of the screen. The bug slowly starts "eating" or biting away the time bar, visually decreasing the time left faster than normal.



How can you distinguish between incoming packets in UDP?

Figure 5-19: Bug Eat Effect

Fake warnings or popups cover the question area.

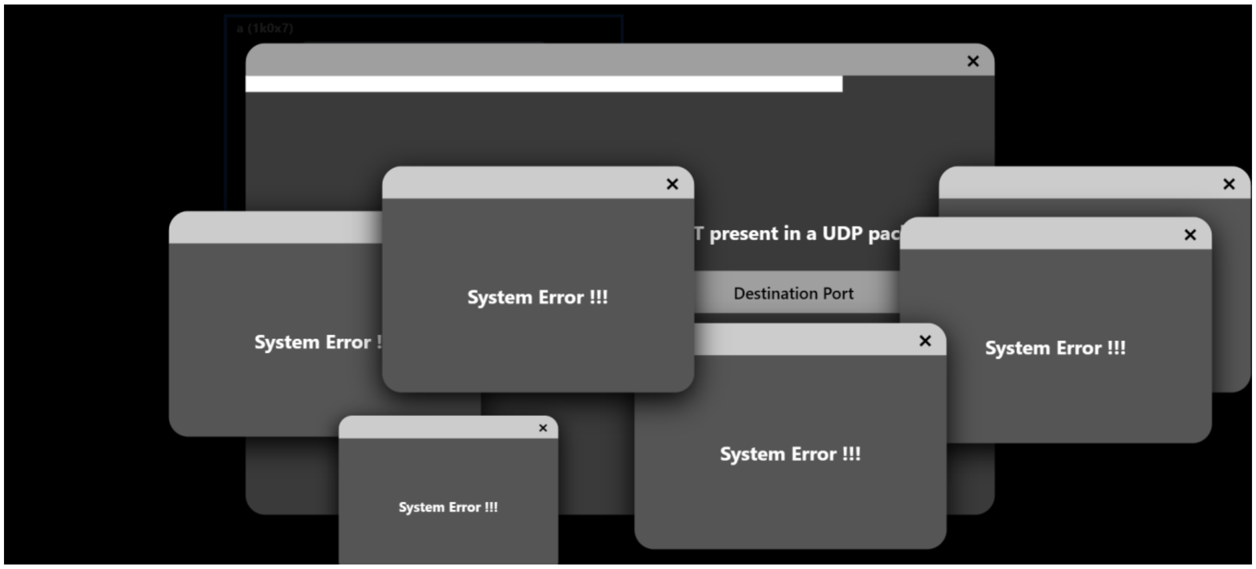


Figure 5-20: Fake Popup Effect

Matrix-style green code falls in front of the question panel. And the content inside will be randomly changing characters in each 3 seconds.

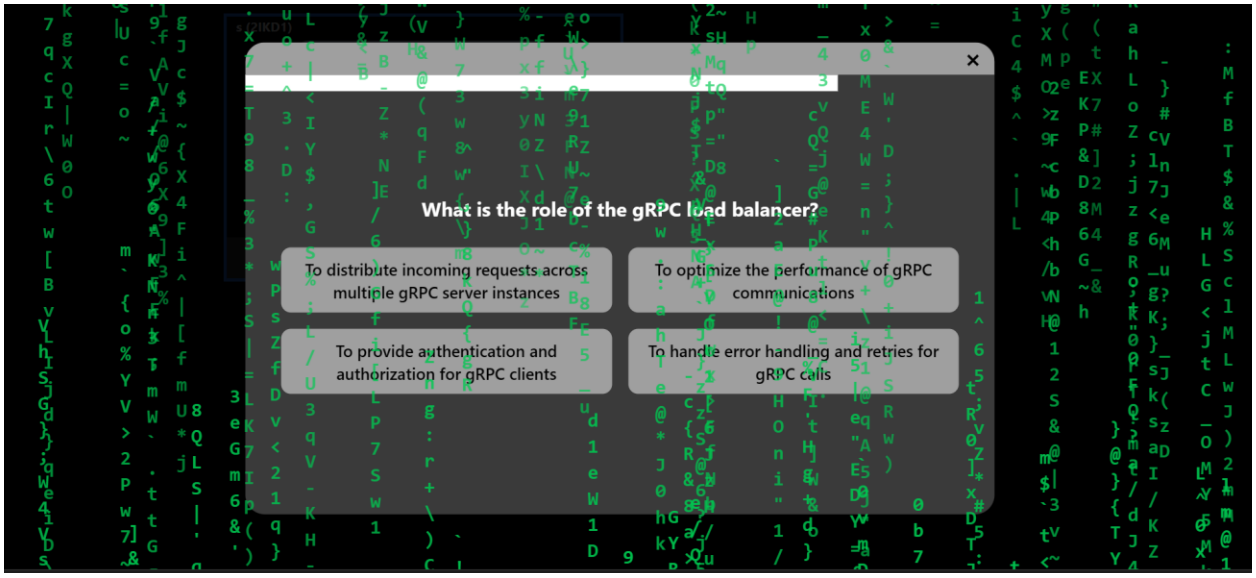
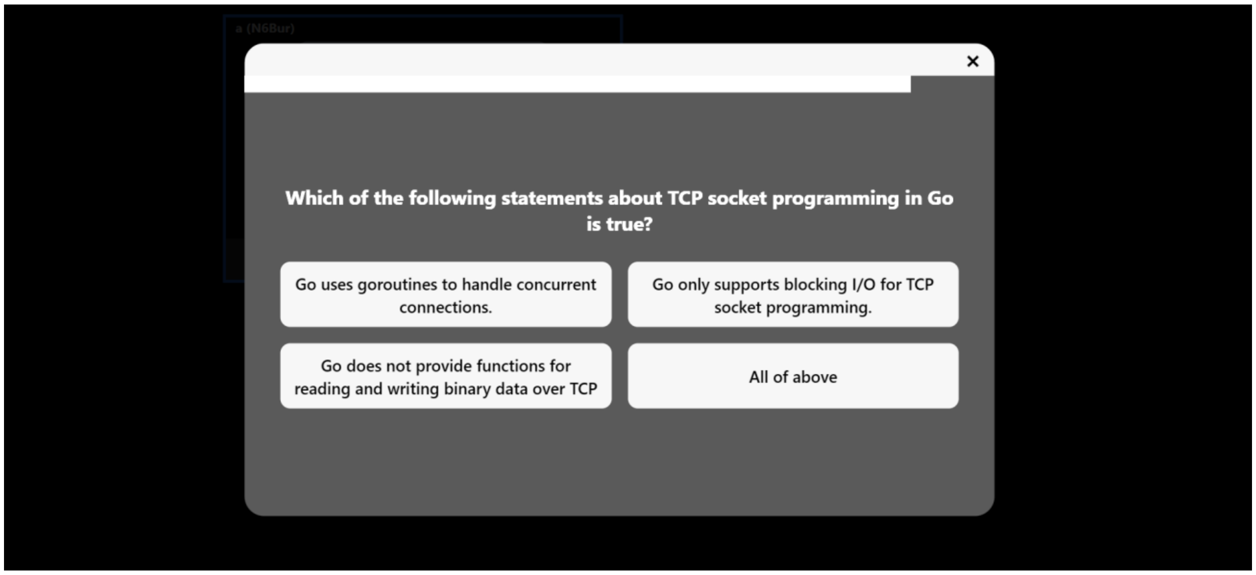

What is the role of the gRPC load balancer?
To distribute incoming requests across multiple gRPC server instances
To optimize the performance of gRPC communications
To provide authentication and authorization for gRPC clients
To handle error handling and retries for gRPC calls

Figure 5-21: Code Rain Effect

The screen blinks continuously.


Which of the following statements about TCP socket programming in Go is true?

Go uses goroutines to handle concurrent connections.

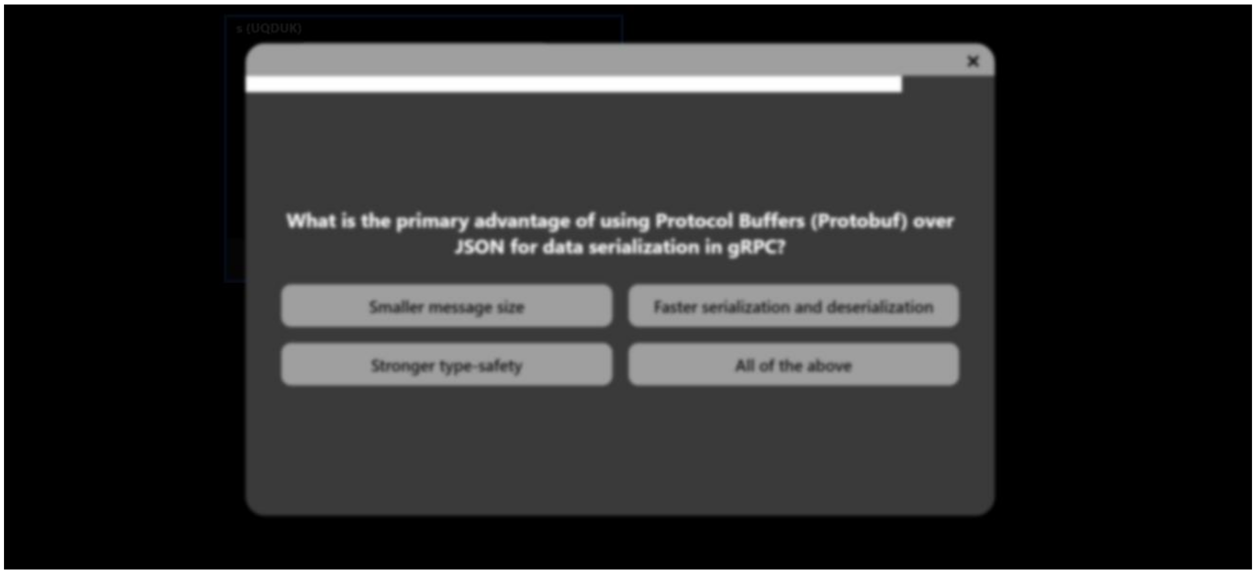
Go only supports blocking I/O for TCP socket programming.

Go does not provide functions for reading and writing binary data over TCP

All of above

Figure 5-22: Flicker Effect

Blur the whole screen to reduce the sight.


What is the primary advantage of using Protocol Buffers (Protobuf) over JSON for data serialization in gRPC?

Smaller message size

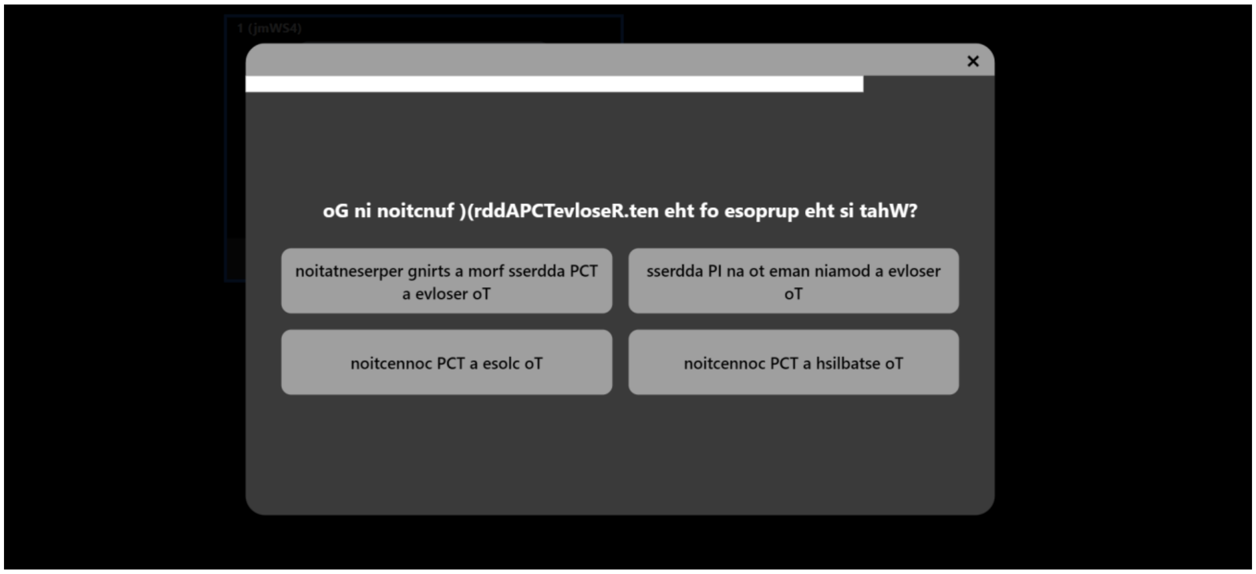
Faster serialization and deserialization

Stronger type-safety

All of the above

Figure 5-23: Blurry Effect

Question and answers appear in reverse order.



oG ni noitcnuf)(rddAPCTevloseR.ten eht fo esoprup eht si tahW?

noitatneserper gnirts a morf sserdda PCT a evloser oT

sserdda PI na ot eman niamod a evloser oT

noitcennoc PCT a esolc oT

noitcennoc PCT a hsilbatse oT

Figure 5-24: Backward Text Effect

When one player runs out of health, that player will disconnect from the room and navigate to the game over screen (Figure 5-25)

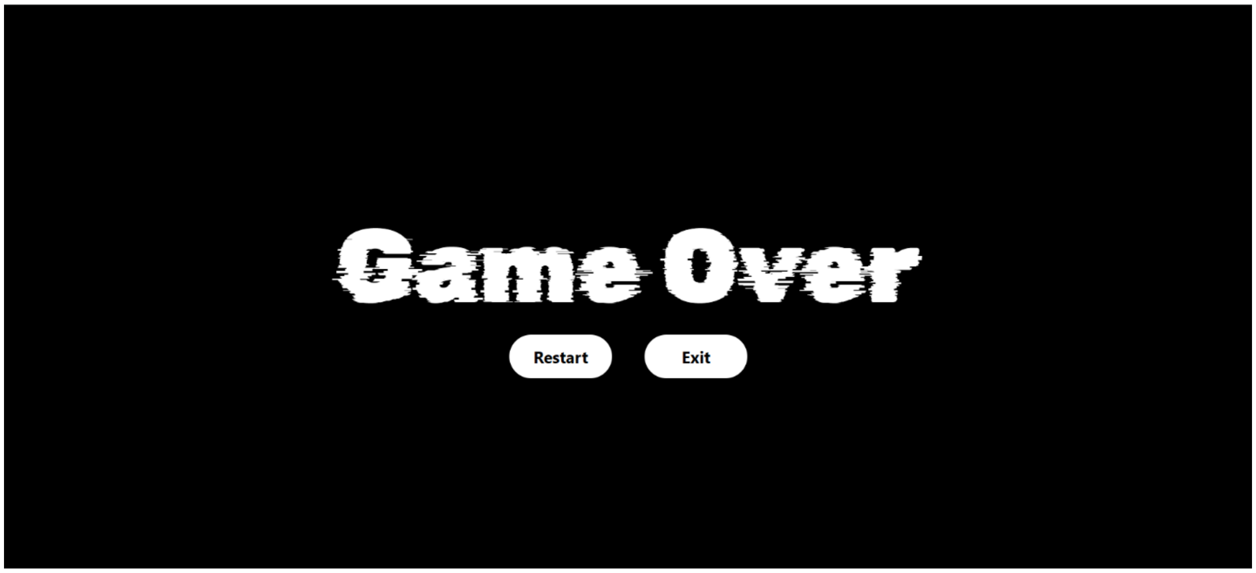


Figure 5-25: Game Over Screen

3. Comparison with Initial Objectives

The initial plans were about making a live-time multiplayer setting in which WebSocket was utilized for the gameplay synchronization, allowing for multiple rooms, introducing sabotage features, the health and elimination logic, and the user interface design.
Withal, the inline list of the following stages was the biggest part of the goals:

- The transmission of real-time data via WebSocket was complete and proved to be successful.
- The game's operation of server-client connection and room creation for the multiplayer environment was in good shape.
- The main basic game logic, such as answering questions and triggering sabotage, was in place.
- A very basic but sufficient UI enabled any player to navigate the game smoothly without any obstacles.
- Tracking health indicators, elimination, and winning conditions indeed fostered the gaming experience with high fidelity.

That was by the time we had hardly any of the advanced objectives:

- The reason for the lack of this was the lack of available time, so the effects of the sabotage were not too impressive
- The question system was completely unresponsive; it didn't have levels of difficulty or any randomization.
- Neither user authentication nor data storage remained unaddressed, so no progress was made here.

Despite this, the system still managed to achieve its intended purpose of real-time gaming, and the strategy set forth for future expansion is quite strong, subsequently including brand new sabotage effects, a broader choice of question mechanics, and user accounts.

CHAPTER 4: DISCUSSION, LIMITATION, FUTURE WORK

1. Discussion and Observations

Bug Brawl was an ideal instance of how real-time multiplayer gameplay, facilitated by WebSockets, can be carried out. The frontend and backend connection was correct, and all connected clients' state synchronization was affected. Since they experienced visual impacts and sabotage, all connected peers were able to participate in the game.

2. Limitations

- User system: Every player is regarded as a guest, and there is no persistent user system. User history and analytics cannot be tracked.
- Database integration: Absence of database integration. Every piece of game data is stored in memory and is erased when the server restarts. Long-term tracking and analytics are, therefore, impossible.
- Mobile optimization: Lack of mobile optimization. On desktop computers, the game layout functions flawlessly; however, on smaller displays, it is not responsive.
- Question system: There is no category or difficulty management, and the questions are hard-coded and static. Additionally, players are still unable to add questions based on the subjects they wish to cover.
- The game does not fully support playing on two different browsers (e.g., Chrome and Edge) on the same device.

3. Future Enhancements

- User profiles and authentication: Add login features to enable ongoing score tracking and individuality.
- Database Integration: Store user information, match history, and leaderboard statistics by connecting to a backend database.
- Mobile Support: Update responsive CSS to rework layout and interactions for mobile users
- Advanced Sabotage Effects: The sabotage effect can stack on each player to make gameplay challenging.
- Better Question System: Use a database to store questions with timed, categorized questions. Making an admin panel for question management.
- Spectator Mode: Let eliminated players watch games in progress.
- Game Reconnection: Use logic to allow players who have been disconnected to re-enter the room during the same match.

CONCLUSION

Bug Brawl certainly was a simple yet great show of a multiplayer game that uses WebSockets. The project helped us build our front-end and back-end skills while also allowing us to observe how real-time systems operate. It was the real experience of the real-time gameplay dynamics that took the experience to a level even higher than secure matchmaking. Provided with additional time and maybe new features and changes, the game will be the one to go for on the client's side.

REFERENCES

1. **React – A JavaScript Library for Building User Interfaces**
   React. (n.d.). Retrieved from https://reactjs.org
2. **Vite – Next Generation Frontend Tooling**
   Vite. (n.d.). Retrieved from https://vitejs.dev
3. **Tailwind CSS – A Utility-First CSS Framework**
   Tailwind CSS. (n.d.). Retrieved from https://tailwindcss.com
4. **Framer Motion – Animations in motion for React.**
   Framer Motion. (n.d.). Retrieved form https://motion.dev/docs/react-motion-component
5. **Gorilla WebSocket – Go WebSocket Implementation**
   Gorilla WebSocket. (n.d.). Retrieved from https://pkg.go.dev/github.com/gorilla/websocket
6. **Gorilla Mux – A Powerful URL Router and Dispatcher for Golang**
   Gorilla Mux. (n.d.). Retrieved from https://pkg.go.dev/github.com/gorilla/mux