Artillery Game Report

by chenxuan xiang

Introduction

For this assessment, I have completed all the requirements on the list. I use react as the front end and node.js/express as the backend. Since it is a simple web browser game and there is a time limit, for simplicity I did not use a database. The server side uses web sockets (socket.io library) to connect to the clients side (react) since the requirement is the motion of two players should be shown. Basically, the idea is to use the "Room" concept to put players in every game instance together and broadcast the game state, and send private messages to certain players. I used localstorage in the web browser to memorize the game instance id in order to achieve reconnection after disconnection.(you can try close one of the browser when the projectile is still flying) For the calculation of the movement of the projectile, I set a time interval to decompose the movement into pieces and calculate over time. In the frontend, I use canvas to draw the game and show the game state. I use react-router to do path control. I deployed the app in Heroku and I also included the source code.

Deployed version of the app: **https://artillery.herokuapp.com**

Backend

In the backend, there are four main classes: GameMaster, Game, Player and Utility.

GameMaster acts like a housekeeper who manages multiple game instances and contains real-time communication with the frontend. It has a map structure (in js it is an object) to store the games. It find empty slot for new player and delete expired rooms.

responsibility:

- Handle the socket connection, CORS problem and mapping between sockets and game instance.
- Handle game instance creation and deletion, keeps track of the active games (in a map structure) and game instance expire timer.
- Help new player find or create a game room or help old players reconnect (including recovering the game state)
- handling real-time communication between the server and clients, mainly sending corresponding game states to the frontend to display.

Game class contains all the info and state needed for the game, including players' position, projectile's position and velocity, field dimension, wind info(wind will randomly change between fires) and expiration timer for this game instance if there is only one player in the game instance for a long time. The most important is the game class will handle the movement of the projectile.

responsibility:

- contains a list of player objects.
- handle adding player, rejoining player, starting the game, resuming game
- handle the core function of this game: projectiles moving and attacking.

Player class defines the basic information about a player, each player is identified with the socket id. Player instance also has a state to indicate is they are online or offline.

Utility class manages the mathematical calculation of the movement of the projectile. The basic idea is given a small time interval, delta_t, I decompose the force into horizontal and vertical directions, and then calculate the acceleration in the two directions respectively, then get the velocity after the delta_t time. After that, I calculate the moving distance according to the velocity and a short period of time. It is a kind of Calculus-biased approach to decompose the movement piece by piece. The frame rate that I used is 60FPS so the delta_t would be 1/60s.

Frontend

For the frontend, I used the react with router. Since time is a little tight, I end up using some not elegant way(use ref to mirror function) to handle the closure problem due to the socket io registration. The frontend has three pages: Welcome page, Game page and End page. I did not do much styling on these pages. I just want this to be a small but complete project. For the game page, I use plain canvas to draw the cannon and projectiles. Besides, the frontend will receive the message from the server and display both players' motion on the canvas.

So a simple user story would be described as follow:

The user connects to the server and then enter the main page which has rules and stuff, then he clicks "start", and the gamemaster will assign a game room for him, there might already be a person waiting in the game room. If the room has two players, the GameMaster will start the game. Then the player can choose the angle and speed of the projectile to fire. The Game instance will take over the job of handling the movement of the projectile and display the motion on both clients' screens. Then if the projectile hit the ground but is not within 3m distance from the opponent, there is an alert to tell how close you are. Then the player can see adjust the angle and speed based on experience and the wind info to hit the target. If the player hit the target, the end page will show "you win!". The player who loses will see "you lose…" and a hint.

Discussion

I decide to store almost all the states and do almost all the calculations in backend. This is because this game requires strict game state synchronization, which is better handled by backend computations and storage. Since I believe that a single source of truth is of great importance for synchronization. Besides, the calculations of this game are relatively simple, the number of players is small, and the game scale is relatively small so it will not be a huge load for the server. What is more, separation can make development much easier, so I choose to separate the frontend and backend where the frontend is responsible for display and the backend is responsible for calculations and synchronization.

When deployed, I build the whole react app and make the backend node.js to serve the build. As for this design, it is completely feasible to separate the deployment of the frontend and backend. I use GitHub to store the code and then link GitHub to Heroku. Finally, I utilized Heroku to help me continuously deploy this app.

Local Deploy instruction

1. go the project directory
2. run 'npm install'
3. run 'npm run build'
4. run 'node server.js'
5. go to http://localhost:8000 to view the result.
6. Remember to open two browsers to battle since one browser will be considered as only one user.

Challenge:

- The whole structure design is not crystal clear to me at first sight. I have to redesign and refactor my code as I write it.
- For this assessment, synchronicity is very important, especially since players can regain their game state like they never disconnect.
- For a relatively short time completing this project requires me to prioritize tasks and solve them by the rank of importance. It also tested my learning ability.

Thoughts:

This assessment was fun for me, but the time is a little tight. Therefore, I did not have enough time to write a very detailed report. However, I would like to give a more detailed explanation or report if you want. Besides, this project involves relatively high synchronization so I spend some time testing the edge use cases like refresh, Influx of many players quickly etc. However, there might still be bugs and inconsistencies, but I do not have any more time to test it. I have considered scalability when designing the structure. However, I think there is still room for improvement.