Multiplayer game implementation documentation

The original idea was to implement P2P networking for a browser based multiplayer game using WebRTC. The implementation includes server code (server.js) that handles the matchmaking and assets, client (client.html) code that handles player input and connections, and host (host.html) code that includes all the game logic and message handling.

Preparations

There is no player limit, separate asset server or version handling. The web browser makes a request to the server that serves the html file for the client. The server tells the browser if the game assets have been updated when the request is made so up to date data doesn't need to be downloaded.

Connecting the players

First the host browser initializes a game by downloading host.html and starting a two-way tcp socket (using socket.io) with the server. The host receives it's random four character code and displays it on the screen so clients can see it.
The clients will download the client.html and connect to a STUN server to receive their descriptors. Clients will initiate connection with the host with the lobby code and start to form WebRTC connections with the data from STUN servers. This part sometimes works and sometimes it doesn't (maybe because of some limits on the STUN servers I used). When the WebRTC connection is formed the client sends a "register" command with its nickname and server responds with "connectionEstablished" so the client knows it is connected. Client can disconnect from the game server and start to send input to the host. When enough players are connected to the host, the host can simply press "start game" to disconnect from the game server and to start the actual game. Other players cannot connect during the game. There is no lobby on the game server, the players disconnect as soon as their WebRTC connection is established to the host.

The game session

Players send their input each 15 millisecond to the host without any redundancy. The packet includes direction and speed. There is no need for player id because the host keeps the WebRTC connections separate and knows which messages come from which client. During testing there didn't seem to be any problem playing without the packet counter so it got left out. The host sends a keepalive message each second to the clients and clients will try to reconnect to the server after three missed messages. The problem is that there is no way to reconnect to the host with WebRTC without the game server so the keepalive is useless.

When the game session ends, there is no “game end” event. Instead, the host reconnects to the server to receive it’s new lobby code so new clients can join. The existing clients will reuse their connections with the host. If client got disconnected from game the new code must be manually inputted to the client form. Because of random bugs in the game and WebRTC code the web page must be refreshed after inserting the new lobbycode.

Conclusions

90% of the implementation time was spent on wondering why the WebRTC connections sometimes worked and mostly didn’t work. Sometimes the clients needed to reconnect tens of times to the server to initialize the connection with the host. Connections fail if I WebRTC is initialized after form submit, I still don’t know why. I thought also using QR codes on the host screen but it appeared that WebRTC needs two-sided handshake process meaning that there needs to be a signaling server no matter what. The whole process was frustrating and I would definitely use a library for initiating WebRTC connections in the future.