**If this is right up your alley please feel free to see the above Github link to check out the entire project including it’s history and source code.**

SpaceRoyale.io is an online multiplayer game supporting up to 50 real time players. The game is built entirely in JavaScript utilizing NodeJS for the server code and in-browser JavaScript via canvas to deliver a front end experience for the players. SpaceRoyale was a collaborative effort between me and a colleague of mine, [Ryan Toole](https://rtoole13.github.io/). Ryan designed the physics systems like collision handling, he designed all of the artwork, and worked with me on game design and generate implementation of game mechanics. Whereas I developed the architecture of the networking code, AWS hosting, database design, and front end webpage to support the game. Many challenges were presented in building this game, but the knowledge and experienced gained from this endeavor were well worth the effort. In this article I will walk through the methodology used to design the game, reveal the architecture of the infrastructure, as well as explain what I learned at the end of the project.

*Methodology*

            From the inception of the concept the goal of this game was deliver a Battle Royale game experience in a shorter timespan than traditionally available in the genre. Selecting a browser based game was important for this timespan as it would allow more casual players to enter without having to install or purchase the game. The goal was to have many quick-join players enter an arena to battle and win or lose within a 3 to 4 minute time frame. A full breakdown of the gameplay design can be seen on our google doc [here](https://docs.google.com/document/d/1e_ROuaIE-R75-oeXf_H0CHULeA28QofNQ7EeoZG2x7I/edit).

*Architecture*

            The game started out as a simple browser based lobby built on the socket.io framework that would allow players to join and move their square around. There was no “game” per say at this stage but it had all the workings needed to support any multiplayer game built into a browser. This stage was forked on [GitHub](https://github.com/sjdodge123/Multiplayer-Game-Demo) and the plan was to have this available to branch into several other online games if desired.

            As the game concept was formed and the need arose, the architecture needed to support the gameplay expanded. The game in its final state supports multiple lobbies of gameplay at the same time. This meant that if Player A joined and started a game when player B joined they could start a game independently and the server could handle multiple sessions occurring at once. This architecture is implied based on the menu structure of the landing page. In order to simplify the player experience the player simply chooses a ship color and name and presses play. The server then will find an available lobby to join and the player enters. Once the starting conditions are met that lobby will then create a game and all players in the lobby will be entered. If the lobby capacity was reached before the game could start a new lobby would be created and any newly joining players would be directed to that lobby. Given that the game server has enough bandwidth to support multiple lobbies this lobby architecture is scalable and could even be reconfigured to hand play sessions over to additional game session servers. This is just one example of the architecture in the game that I built.

*What I learned*

            From the beginning I knew this was going to be an ambitious project. I had never made a real-time multiplayer game and I wanted to understand the efforts of what that would take. In my conceptual learning phase I needed to understand how data could be transmitted in real time across the internet to my application on multiple computers. I began my research into sockets. After many hours I was able to produce a flash based (Actionscript3) demo that could move an object on a server from a client. I had hosted a game server on my home computer, set up my router to port forward properly and drove to a local Starbucks to test it out. That first time I moved my character on the screen was amazing! I proved that I could achieve the basics the rest was about scaling.

            Later in the project development I switched to a different framework for better native support of my application as well as drastically changed the scope to be something more palatable. Using Sockets.io I did a fast demo provided on their website on setting up an instant message chat client. This was amazing as I realized its potential as a web framework immediately. I could bypass having to deal with sockets directly and let the framework handle the dirty work.

            In the original design of the project I wanted to learn how JavaScript can interact with databases to store basic player information and allow authentication for the sake of player progression. I had years of experience working in ServiceNow to leverage which is essentially a giant relational database with a JavaScript script base. Knowing how the database should be structured, I went off and learned how to make a Maria database with AWS. I connected this database to the game to store quests/player information/kill counts etc. Once I understood how to connect these things on a native level I feel more confident using existing frameworks that interact with these tools.