ECE 493 Capstone Design Project
PolyPong Final Report
Date: Friday, April 16
For: Dr. Scott Dick
By: Micheal Antifaoff

1. Discuss the problem domain for your project. What is it, and why is it important? Be specific.

PolyPong is a multiplayer online game that recreates the original Pong game with modern technology and the ability for multiple players in different locations to connect and play the game online. No software download or installation is required – all you have to do is visit PolyPong.ca. Our game was built from the ground up with connection and fun in mind. While PolyPong is similar to the original Pong in terms of gameplay, we have added eleven powerups, many different coloured paddles and a global leaderboard to enhance our game as well. Users can play as a guest, or they can register and unlock the ability to earn XP, to earn different coloured paddles and to participate in the global leaderboard ranking PolyPong players from all around the world. With a clear understanding of what our game is and isn't, the problem domain of the game becomes easier to analyze. PolyPong resides at the intersection of three different domains: gaming, distributed systems (real-time synchronization across geographically separate clients) and online social experiences.

The first and most obvious domain which PolyPong belongs to is gaming. Gaming is an incredibly important domain in its own right, with video games generating an astonishing US$134.9 billion annually worldwide in 2018.[1] Games are important because they take us out of the mundanity of everyday life through the power of play and fun, opening up new worlds for us to explore and discover. Games exist for players to play and enjoy; as a result, one of the most important non-functional requirements for PolyPong is that it is fun. The core game mechanics need to be solid: paddle movement should be responsive and speedy; gameplay should be quick and easy without being too fast or overwhelming. It is hard to capture the essence of fun in words – you simply know it when you pick up a game and are enjoying it. What is tremendously interesting is that fun is not created equal – what some enjoy, others find to be toil or work, and what still others enjoy, some find to be too simplistic or easy. It was definitely a challenge to make PolyPong fun to play, and without shaped paddles, we did find gameplay could become repetitive, especially in a 2 player setting where there are no real challenging angles and so the only challenge is the speed of the ball. We did our best to make PolyPong fun by adding a sense of progression to the game (through XP, leaderboards and unlocking different paddle colors) and by adding novelty through the form of powerups, so that no two games are ever alike.

The second problem domain for PolyPong can be seen by viewing PolyPong as a distributed system, with the server at the heart of the system and each client representing one node of the system. Since PolyPong is a real-time game, the clients must all be relatively in sync in order for the game to be enjoyable to play. Since paddle movement, collision detection and game over handling is done individually on each client, time synchronization is another difficulty we faced. Latency places a big role in this – the greater the latency, the more out of sync the game can become. Time synchronization is important with PolyPong so that each player is viewing and playing a game that is very similar, if not identical, to their fellow players. This places PolyPong within the domain of distributed, synchronized systems that work even when the clients/nodes are not geographically close to each other. This is a tremendously important problem domain because distributed systems are becoming increasingly common and they allow for us to make use of all available resources. Today, more people own a computer with an internet connection, a decent processor and a reasonable amount of storage than ever before. As more of these devices are bought, the amount of computing power in the world is steadily increasing and being able to tap into unused compute resources (through distributed systems) can allow us to make progress on a number of hard problems which require extensive compute power (such as weather forecasting, protein folding or any computationally hard/big data problems which can be broken up and parallelized). While PolyPong is not an intensive game to run, it does embody some of the principles associated with distributed systems, and also with synchronizing these systems in real time. For example, PolyPong offloads a great deal of work from the server to each individual client (including paddle movement, collision detection and game over detection and handling). In fact, PolyPong's server largely exists to maintain an "authoritative state" of the game and to handle any update messages from one client that need to be broadcast to the other clients connected to the game on the server. Each client embodies the principle of horizontal scalability (since computing happens independently on each node), while the server facilities all communication between the clients to ensure that the clients are synchronized in time across geographic distance.

The last domain which PolyPong resides within, which is closely tied to the first two domains, is online social experiences. As technology has improved over time, fast, low latency internet combined with high-speed processors and abundant memory has ushered in an era of online spaces where people can come together and have virtual experiences in a shared, online space. This has been a noticeable trend for some time, but with the advent of Covid-19, the importance of having online shared spaces has become abundantly clear. These sorts of online spaces have replaced the parks and coffee shops for the next generation and provide social interaction to help foster connection between individuals. On a large scale, examples of this could include Fortnite [2] (the recently added Party Royale mode is an excellent example of an online social space [3]), custom Minecraft servers, Animal Crossing [4], or even Club Penguin (which was one of the first commercially successful online social spaces where friends could hang out, chat and play minigames [5]). Online multiplayer video games have increasingly become recognized as social spaces as well. [6] While PolyPong does not achieve anywhere near the size and scope of the previously mentioned games, it does fall within an identical domain: friends, or even strangers, who are separated in distance and potentially even time zones, coming together to share an online experience. The major challenge within this domain remains creating an experience that people can enjoy while physically separate but online together. This domain remains an interesting and important space; the development of VR and AR, similar to the OASIS in Ready Player One, also appears to be heading in the direction of online, shared social spaces. For evidence, one needs not look further than two of the biggest names in tech: Microsoft is building Microsoft Mesh, which provides social spaces people can interact within (but more as an enterprise-targeted offering, probably due to the $US 3500 price of the HoloLens 2) [7], while Facebook remains focused on developing Facebook Horizon, which provides spaces for friends to hang out within but also allows for users to create their own custom environments using 3D shapes and templates. [8] PolyPong represents an entrant into this domain because our game is designed to be played with friends online, letting players share time and a unique experience with friends.

2. Discuss existing solutions. What products are already in the market in this domain, and what features do they offer?

In terms of Pong as its own domain, a number of other versions exist online or as downloadable apps, but none that I was able to find replicate our exact functionality. Some of the more interesting ones include:

- https://pong-2.com: An online version of Pong with single player (against a computer-controlled opponent), multiplayer (two users on the same device) and online multiplayer (two users on different devices). This version only supports up to two users and does not have powerups, accounts/XP/skins, or any leaderboard functionality.
- https://playpong.net: Contains 9 different versions of Pong, all of which are single player against a computer opponent, or two users on the same device playing together (no online multiplayer). None of these versions support more than two users and none have powerups, accounts/XP/skins, or any leaderboard functionality.
- https://github.com/SourKream/PingPong: Online multiplayer with 2 players only, but you need to install Java and specify your friend's IP address (as opposed to it being hosted on a website with no software installation required). This version only supports up to two users and does not have powerups, accounts/XP/skins, or any leaderboard functionality.

While these are just three examples, Pong is video game cannon and hence many, many programmers have written their own versions of it at some point in their career. I was unable to find any version that supports more than four players in an online multiplayer mode (ours currently supports at least 12 players simultaneously). Many versions have implemented two player online mode, some even have powerups that are different from ours, but I was not able to find any with the ability to create an account, much less with an XP/Leaderboard/Skins system (which gives players a sense of progression). One feature that other games had which would be nice to implement is computer-controlled opponents so that if you had four friends but you all wanted to earn more XP, you could just add in 5 or even 8 computer controlled opponents and then start playing. We also wanted to implement a labelling system (so that you could see the usernames of the people you are playing), along with a chatroom. None of the multiplayer Pongs that I came across had labels associated with the paddles (since none had an account creation system and hence no usernames), but one version did have a chatroom (which would have been fairly straightforward to implement and would be a nice feature. Unfortunately, we did not have enough time and it was not a functional requirement for us.) Another feature that other versions had was offline play; this would be possible to add to our game in single player mode, but since we wanted to focus on online multiplayer with your friends, being connected to the internet is necessary to play PolyPong.

In the larger domain described in Question 1, the intersection of gaming, distributed systems (real-time synchronization across geographically separate clients) and online social experiences, there are a number of existing solutions: Fortnite, Minecraft, Roblox, Animal Crossing, Club Penguin Rewritten, Facebook Horizon, online Monopoly, and many more. The majority of these titles would fall into the category of Massively Multiplayer Online (MMO) games. It is harder to make a direct comparison between our product, PolyPong, and these other solutions since they each address slightly different aspects of online social gaming distributed systems; also, the systems mentioned above have more resources and people attached to them and can, as a direct result, afford to implement some nifty features we did not have the resources to implement (both in terms of time and headcount). However, clear similarities can be made at once: all of these games maintain a real-time state of the game and do their best to make the state consistent across all clients. In fact, the largest difference between PolyPong and many of these MMO games, aside from game mechanics and graphics, is the scale of the game and the experience of the companies with making games of this scale. As discussed, PolyPong has elements of horizontal scalability built in, but since we are only running one server at the moment, if we wanted to scale our game we would need to add significant functionality in terms of load balancing across multiple servers. This is just one example of the features present in MMO games that PolyPong does not have. Since this was also the first team any of us had built a game which offers online multiplayer, we learnt many things along the way which would have been useful at the start of development. As you might imagine, some of these companies have made 5, or even 10 successful, well-played online multiplayer games (such as Epic Games, the development company behind Fortnite). We, as developers of PolyPong, are only beginning to scratch the surface of their wealth of experience and knowledge.

3. Discuss your solution. What design choices did you have to make? What were the alternatives you considered? How is your solution different from the existing ones?

One of the first design decisions we made came out of a meeting we had immediately after receiving multiplayer Pong as our capstone. We discussed how we wanted our game to be distributed, and we realized that we wanted the game to be available to as many people as possible. We wanted players to be able to start playing our game in seconds, not minutes or hours, and so when choosing between a web app and a more traditional software application that had to be downloaded and installed in order to be used, we went with a web app. This meant players only needed to visit our website in order to be able to play our game.

With that crucial decision out of the way, it was time to choose our technology stack. In the beginning of this project, I had never built a web app before. I had never used Deno (or Node.js), Svelte (or React), or any of the other technologies listed below (save for MongoDB). I only had a vision for how the app would look, work, and feel, while Arun, who has built multiple web apps before, had the knowledge of which technologies could actually bring such a vision to life. As a result, many of the design choices we made early on were made by Arun, since he had the most experience with web development. We discussed all of these choices together as a group, and my principal role in these discussions was to ask questions and tease out how we could use the different technologies available to build the app we dreamt about. Arun's deep understanding of the benefits and drawbacks of differing technologies really helped our group out because I could explain a feature that would be cool (for example, link-based game joining) and Arun would think about it, and then explain which technology would be best for that use case and why. In the initial design phase of the project, my job was to ask lots of questions and make sure we could do everything we wanted to do with the technologies we were selecting.

We then began to think about the overall structure of our application as a distributed system: How would each player's device communicate with the other devices? How we would maintain a consistent state across all devices? At this point, we considered two alternatives: a client server architecture and a peer to peer architecture. We went with a client-server architecture because we wanted an authoritative state of the game maintained in a centralized location that all clients had access to and could provide updates to. We also needed to keep track of user statistics and XP earned to satisfy our functional requirements, so having a centralized server made the most sense and seemed to be the easiest approach. Once we decided to go with a client server architecture, we began to think abnout how work would be distributed within the system, and we began to think more and more of our system as a distributed system. We realized that by supporting up to 12 players, it is crucial that we do as much processing as possible on each client, offloading work to each client and freeing up the server to handle as many simultaneous games as possible. We choose to do this by handling all paddle movements, collision detection and game over/game winning functionality on the individual clients, and getting the clients to update the server whenever their state changes. This lead to one other important design decision, which involved the choice between WebSockets and WebRTC. Since each client was responsible for updating the server whenever its state changes and since each update is crucial to the flow of the game, we went with WebSockets, which makes use of a reliable TCP connection where packets are guaranteed to be received in order as opposed to UDP (where packets are not even guaranteed to be delivered). Our game did not need the higher performance of UDP but did need the reliability of TCP, so WebSockets were an easy design choice.

With much of the architecture of our game laid out, we began to choose our working languages. For a frontend language, we chose TypeScript because it compiles to JavaScript, which runs in almost every browser and is fairly speedy. We went with TypeScript over pure JavaScript for the static typechecking performed by TypeScript, which helped us over the course of the project to avoid many easy-to-make errors caused by improper type conversions or undefined/null objects. We went with TypeScript over a WebAssembly compatible language(such as Rust or C++) because of simplicity and support. TypeScript was simpler because Rust and C++ require memory management, while TypeScript is garbage collected – we don't need the performance offered by manual memory management because our game is not tremendously intensive and we wanted to avoid unnecessary complexity. TypeScript has also been around for longer than WASM. WASM represents a relatively new standard (2018-present) and would have a larger learning overhead since it does not yet have an abundance of documentation. For a backend language, we decided to also choose TypeScript. The main reason we TypeScript over any other language (which may have been faster/had better libraries/etc.) was for continuity between frontend and backend code. We wanted consistency to allow the for simplest possible communication between frontend and backend; with both frontend and backend code written in TypeScript, our workload greatly reduced because we could use the same Game object (as well as Paddle, Ball and Player objects) across both frontend and backend code.

In terms of the framework on the frontend, we chose Svelte over React. Svelte was easy to learn and worked well with a mockup I had developed in HTML, CSS and pure JS – since we were able to port the mockup over quite easily, we decided Svelte was the framework for us. In terms of the framework on the backend, we went used Deno w/Oak as our routing library (over Node.js w/Express.js). We chose Deno because it was created by the same person who created Node.js (Ryan Dahl), and contained improvements from Node.js without additional complexity. Deno included more security out of the box, which helps our game to become more secure by default. In terms of hosting the frontend, we went with Cloudflare Pages (free) over Github Pages (also free) or Netlify (also free). There was no specific reasons we chose Cloudflare pages, other than Cloudflare's immense distribution network; any of the providers would have accomplished our goal. In terms of hosting the backend, we chose Cybera which was free over Digital Ocean (paid) or Google Cloud Platform (paid). We liked Cybera because it is free for students and local (operates in Alberta), so we did not have to abide by COPPA or GDPR or California's privacy regulations. We packaged our backend in a Docker container for easy deployment, and we use a Dockerfile so that we can easily reproduce the backend deployment over different hardware/OS configurations. We went with Docker over Kubernetes because we found Docker to be a simpler solution.

In terms of account sign up and log in, we considered many different options. At first, we were planning to use email authentication (similar to Slack). However, we decided to go with OAuth after meeting with the professor and discussing as a team how the login flow would look (we wanted players to be able to login without having to open their email every time). We also considered developing our own username and password database, but quickly eliminated this option after realizing that additional passwords are a hassle for the player to remember and a hassle for us to secure. Since we didn't need to worry about passwords, our database could be extremely simple. All we needed was a simple and fast database and we chose MongoDB over a relational database such as SQLite because of MongoDB's support for the JSON format and its relatively simple commands.

Finally, the last but perhaps most important design choices we made were about what features our game should have, and how the game should look, work, and feel. The most important feature of any game is that it is fun to play, and fun comes from progression and improvement, as well as novelty. To give players a sense of progression and improvement, we decided to have an XP system that allows them to earn 1 XP for each player they defeat while playing. As players earn XP, they unlock different colours for their paddles and they are able to see how they rank compared to others in the world while viewing the top players in the world on a global leaderboard. These ideas for creating a sense of progression and improvement come both from other video games we have played, as well as the karate system of awarding different coloured belts as people progress. We envisioned different paddle colours having an associated status element to them, which is why it is important that all players can see each other's skins. To create the novelty required for fun to flourish, we added 11 different powerups to the game. This helps to ensure that no two games are ever alike, since different players will always select different powerups, making each new game a fresh and exciting experience (imagine if 12 players all selected the add ball powerup!). With fun designed into the core of our game, we also discussed whether our game would have a monetary aspect. We all agreed that we dislike ads and dislike games which are pay-to-play (games where buying things can make the game easier for individuals). Our intuition was that paying money to unlock something that made the game easier, or even more interesting would not enhance the game for all players. Since we have the luxury of not making PolyPong our living, we did not feel there was a need for monetization. Our last design decision was about the theme of the game. We decided that since Pong is the original video game, it would be both fitting and nostalgic for our game to take on a retro feel. We used this retro theme as our north star, and everything we implemented had to maintain continuity with the rest of the retro theme. Based on this, we chose a simple color schemes (two main colors – that is it) and a square and blocky design (all buttons are square, the font is pixelated, and even curved/bumpy paddles are retro themed). What we found most enticing about the 70s, when Pong came out, was the impressive simplicity of the initial game. The technology to do all kinds of fancy stuff hadn't yet arrived, so everyone could immediately understand and enjoy what was going on. Without hidden submenus, hard to understand features, or obscure options, everybody is included. This is what we decided PolyPong was all about.

4. Discuss the potential impact of your project on society and the environment. Identify all stakeholders and the project's impact (positive or negative) on each of them. Reflect on any environmental impacts (e.g. Facebook's carbon footprint could be argued to include both its own servers and the power consumed by devices connecting to it).

PolyPong could have multiple dimensions of impact on society and the environment., but the true impact on both will likely be minimal given the saturated nature of the gaming landscape (unless the game was to go viral and user growth skyrocketed). With that said, we see Polypong as having a potentially significant positive and negative impacts on society, as well as a largely negative impact on the environment.

In terms of the positive societal impact PolyPong is a great game that’s fun to play that can bring people together from around the world. For myself, it has been fun to share and playtest PolyPong with both my friends in Edmonton and my friends in other cities. It has provided both my friends and I with some joy and brought us closer through the power of play and fun during what has been one of the most challenging years on record. If PolyPong was to go viral, it would bring that same joy and feeling of connection to players worldwide, which is definitely a positive societal impact of PolyPong. However, like all electronic games and devices, although PolyPong allows for a unique experience between friends, it does encourage the players to spend more time playing PolyPong and less interacting with those around them. Viewed through this second lens, PolyPong could act as a form of escapism and also have a possible negative societal impact. We hope that this is not the case, and that our game solely brings people together in a playful, online social environment. We have created an environment for friends to have fun playing each other, and believe this will have a positive societal impact as the number of people playing PolyPong grows.

In terms of the environmental impact of PolyPong, since the game needs to be accessible over the web, our game is hosted by Cloudflare on the frontend and Cybera on the backend. The power consumed hosting PolyPong is the most direct environmental impact that PolyPong will have, and this is determined by the type of electricity used to power Cloudflare’s and Cybera’s servers. If Cloudflare and Cybera are powered using fossil fuels or natural gas, PolyPong will have a direct impact on carbon emissions (since if PolyPong was not hosted on a virtual server, that electricity would not be used), however if they are powered using renewable energy, such as solar power or wind energy, then PolyPong would have a much smaller impact in terms of carbon generation. As of writing this document, it was not possible to discern Cloudflare’s or Cybera’s power generation method and environmental impact. However, Cloudflare does use a variety of methods to reduce its environmental impact, detailed here [9], and also buys carbon offset credits to offset the power their data centers and offices consume. One interesting finding about Cybera is that they support green tech solutions through data science. [10]

PolyPong’s different stakeholders include guest users, registered users, the ECE 493 teaching team (as project owner), the developer team, and the hosting companies (Cybera and Cloudflare). For both guest and registered users, PolyPong provides joy through playing the game, but could distract them from their everyday activities. For the ECE 493 teaching team, PolyPong will hopefully provide lots of joy during playtesting, but also adds another project that they need to mark. Hopefully, the playtesting is so much fun that it offsets any negatives that result from the workload of marking an additional project! For the development team, PolyPong had a net positive impact, leaving a lasting good feeling of progression and accomplishment. Unfortunately, PolyPong also resulted in many sleepless nights of development but all in all, the positive impact of learning and growing as developers and people outweighed any negative impacts. Finally, PolyPong likely had a small negative impact on Cybera and Cloudflare since we are using their services but currently not paying them, wearing out their servers without financial compensation. This will remain a very small impact on Cloudflare and Cybera unless our game suddenly takes off and goes viral.

5. Describe your exact role in the project, the roles of each of your teammates, and how these roles overlapped. After reading this section, I should have a complete picture of what each team member did, and how these individual roles combined to complete the project.

Arun’s role in our project was incredibly significant. Without his knowledge and experience, PolyPong would not be what it is today. Arun did a tremendous amount of work, largely on the backend, in order to lay the foundation for our game. Since he had experience with building web apps before, he knew the different technologies we could use, as well as the benefits and drawbacks of each one. As a result, he was very crucial to getting our initial technologies chosen (both for frontend and backend) and the general framework of the app developed. Whenever we began to use a new framework, he would always be laying down some boilerplate code which demonstrated how we could use the new framework. Arun’s deep understanding of the many different options we had was essential to our team before the project had even begun.

Once we began to develop our app, Arun’s role only grew in significance. He was responsible for authentication (using Auth0, getting it set up to work, authenticating JSON web tokens on the backend), for getting our server sent up to receive messages and respond accordingly, for being able to broadcast from the server to all connected clients, for the 2-level lobby structure of our backend, and for getting WebSockets set up on the frontend so that the frontend can communicate with the backend. He also worked heavily with our database, getting it set up and writing a majority of the functions which got data from or added data to our database. He also did a majority of our tests, writing all of the tests except for the eleven tests involving the powerups. Finally, Arun made PolyPong accessible for the world to play by registering polypong.ca as our domain, administering DNS records, deploying the frontend to Cloudflare pages, writing Dockerfiles, bundling our app into a Docker container, and deploying the backend to Cybera. Arun’s contribution to PolyPong cannot be understated and he was an incredibly important member of our team.

Josh’s main contribution was working on our synchronization algorithm (the “Dead Recknoning” algorithm). In essence, each client extrapolates the behaviour of other clients and tries to predict what they will do. When an update is issued by the server, the clients’ update their extrapolated verions to match the authoritative state of the game provided by the server. Josh also provided the initial code for the Add Ball powerup, which Arun, Josh and I pair programmed to debug and develop further into the final version of the powerup.

For my role, this was my first time ever building a web app, and the list of technologies I had never used before this project is quite long. As a result, this was an excellent learning opportunity for me and I embraced every minute of it. My main roles within the project were to design PolyPong, build the game, and add powerups. This is a summary of everything I worked on during this capstone project, including whenever I was pair programming with Arun or Josh.

I was the driving force behind the creative vision for the game. In the early weeks (February) I worked on initial mockups for how our game would look and feel so that I could learn HTML, CSS and some JS (roughly 1000 lines of code) I started to work on these so we could get an idea of what functional and nonfunctional requirements we would need to implement (this was before we had submitted our SRS). These mockups demonstrated what the lobby would like, what powerup selection would look like, what leaderboards would like – they captured an initial (if non-functional) version of our game and provided a framework for us to build on, and these mockups ended up being what our app looks like today. I then worked with Arun and Josh to complete our SRS and our preliminary design plan (any and all documentation).

Once documentation was done, my focus switched to learning the canvas: starting with drawing simple lines, then connecting them to form polygons (our game board). I then worked on drawing paddles (one line per side) and creating a simple algorithm for drawing paddles that required us to only keep track of one number (the x-value) per paddle (to minimize the amount of information that needs to be sent between clients and the server). Then, I added the gameLoop(), which allowed us to move the bottom paddle across the screen using the arrow keys (updating state on arrow key press/release) and drawing the updated state in the render() loop. Once a polygon and paddles were drawn and the paddles could be moved, I added a ball that did basic paddle collision detection, then added the ability for us to detect when the ball was out of bounds for the current player (and hence the current player had lost). So far, none of this work interacted with the server in any way, so only the paddle at the bottom of the screen was non-static (you could move it left and right, and it did collision and game over detection). Because each client was responsible for handling its own state and then updating the server whenever its state changes, this was the foundation for our game client-side. In fact, because of how the code was designed, often adding powerups would only require 5-10 lines of code client-side (for example, for invisible ball, we would just be sending an update to the server and wrapping the drawBall() function in an if statement which checked to see whether or not the ball should be visible). At this point, we now had a relatively robust and easy to understand game architecture. Now it was time to work with my group members to broadcast messages to the other clients so that when the ball or a paddle moved on one client, that was reflected across all clients.

With the core game elements done client-side, I then worked with Arun to get a ball moving across the screen consistently across all clients, and to get paddles moving consistently across all clients. We laid the foundation for the message-passing going on over WebSockets between the clients and the server and for keeping the state of the game consistent across the server and all clients. Arun and I worked together on this because he had already done a ton of work on the backend to make this all possible, and I understood the drawings and rotations going on in the canvas, so we were able to get this done quickly. Once we had a functional game consistent across all clients, Arun then worked on adding a lobby system for getting an initial game started. As soon as Arun completed this, we then needed to restart games on player elimination. This became my focus for two or three days and is when I started to become more comfortable with the backend code and how it works. I touched a significant amount of both the frontend and the backend code getting games to restart without the eliminated player

Now that games were restarting, we needed a way of giving players a quick break from the action, so I implemented an animated countdown (the 3-2-1 you see in orange while playing). Around this point, Arun and I began to notice that we had some race conditions in our game where one client would overwrite another client's update with old information, causing paddles to not move properly across the screen. Working with Josh, all three of us pair programmed to try and fix the problem. Josh worked on it for a few days while pair programming with Arun and

I, implementing the Dead Reckoning algorithm for synchronization across the different clients, allowing each client to smooth out paddle and ball movement. Arun and I also worked on the leaderboard and skin selection/displaying skins in-game together, and we both added some REST API calls for stateless communication between the client and the server.

Finally, with the our game working reliably well, I began working on implementing the powerups. I implemented 10 powerups client-side on my own (Bigger, Smaller, Bumpy, Curved Inwards, Curved Outwards, Self Invisible Paddle, Others Invisible Paddle, Invisible Ball, Distracting Background and Trace Ball Path) and pair programmed with Arun to fix powerup synchronization bugs for three of the different powerups (Smaller, Others Invisible Paddle, and distracting background). Josh wrote the initial code for the Add Ball, and pair programmed with Arun and I to complete the ‘Add Ball’ powerup. At this point, we had completed all of the required functional requirements and the remainder of our time was spent writing tests (80% Arun, 20% Micheal), adding comments to our code, ensuring all required documentation was complete and fixing any bugs we discovered.

In conclusion, I worked on a great majority of the frontend and also a significant portion of the backend. On occasion, Arun and my role’s overlapped, and neither of us hesitated to do work when we saw that it needed to be done (whether it was backend or frontend work). This capstone was a ton of fun and I learnt an immense amount along the way – lessons I will take with me into any future project that I work on.

References:

1. https://web.archive.org/web/20190509014637/https://newzoo.com/key-numbers/

2. https://qz.com/quartzy/1493147/fortnite-a-social-space-like-facebook-and-skateparks-once-were/

3. https://screenrant.com/fortnite-party-royale-guide/

4. https://techcrunch.com/2020/05/02/virtual-worlds-video-games-coronavirus-social-networks-fortnite-animal-crossing/

5. https://mashable.com/2011/12/13/club-penguin-disney/

6. https://www.psychologytoday.com/us/blog/video-game-health/201901/video-games-are-social-spaces

7. https://www.theverge.com/22308883/microsoft-mesh-virtual-reality-augmented-reality-hololens-vr-headsets-features

8. https://www.theverge.com/2020/8/28/21405249/facebook-vr-space-horizon-access-public-beta

9. https://blog.cloudflare.com/the-climate-and-cloudflare/

10. https://www.cybera.ca/cybera-uses-data-science-to-support-green-tech-solutions/