ECE 493 Final Report

Arun Woosaree

April 15, 2021

1 Problem domain

The goal of our project is to provide a fun multiplayer social game that provides an experience similar to Pong, but is easily accessible with modern technology. Imagine you have a bunch of friends that want to play a game together, but you all are physically separated. Furthermore, your friends do not want to install any new software on their computers. You and your friends each have a computer with either Firefox or Google Chrome installed on it and internet access. If you ever find yourself in this situation, this game is for you! Not only does our project provide a Pong-like experience, it also adds extra features which adds a unique spin on the game for extra fun. More than two players can participate, and also there are eleven fun power ups to choose from. Users can earn experience points, unlock skins, and compete for the most experience points with the leaderboard feature. Unlike most other games, there is no software to download or install. Most people already have a modern web browser like Firefox or Chrome. All a user needs to do is to visit polypong.ca and start playing. The experience is as frictionless as possible. Users are not even required to sign up before playing a game. Users can enjoy the game as a guest, and sign up later if they choose to.

2 Existing Solutions

A quick google search for "multiplayer pong" reveals that there are a plethora of online projects which already exist that allows one to play a multiplayer game similar to pong. Here are a few examples:

- `https://pong-2.com/` This has multiplayer online play, a single player mode, offline play, online multiplayer, and the ability to change the colours of the baddle, background and ball. However, this does not have power ups like our game. Furthermore, the online multiplayer feature is restricted to two players, while we have no such restriction

- `https://play.google.com/store/apps/details?id=com.AvidGames.Pong&hl=en_CA&gl=US` This requires an Android phone, and an app download. Also, it does not seem to provide online multiplayer with more than two players at once like we do, nor does it appear to have power ups.

- `https://github.com/pstefa1707/multiplayer-pong` This also seems to have a single player mode and online multiplayer. Like the other examples, there is no option to play a game with more than two players simultaneously.\

- `https://github.com/HarshdeepGupta/multi_pong` This appears to work with up to four players, while our game allows for more than four players to play in a single game. Also, it requires Java to be installed, while our game just runs in the browser.

- `https://github.com/Eisah-Jones/Multiplayer-Pong/tree/master/Multiplayer-Pong` This one seems to have a chatroom feature, which we do not have. However, this game seems to have a limit of four for the maximum amount of players that can play at once. Also, this game needs to be compiled, or a binary needs to be downloaded before it can be enjoyed, and the setup is not as simple as just going to a website like our game.

  A common trend among these examples seems to be that most of them have the limitation that the games usually have a set maximum number of players. In our game, we have no such limit, and 6, or even 12 players can easily play a game together. (The game does get more and more chaotic as more players join, and at a certain point, it does get impractical to play a game). However, we did not implement any single player mode in our game, while other alternatives usually have a single player mode with a computer opponent controlling the other paddle. I also found some games that seem to work offline, while our game requires the player to always be connected to the internet.

3 Our Solution

We wanted our game to be easily accessible, so that users can pick up the game, and start playing it in just a few seconds. This helped us narrow down our choices for what programming language to write our application in: The best way to make our application work on almost any device with next to no setup is to make it so that it can just run in a browser. Most modern devices have a web browser based on Google Chrome of Firefox pre-installed. So, if we make a web application, the user does not have to download any new program, or install anything. They can just visit a website and start playing right away!

The most popular language for web applications is JavaScript, however, we chose to go with TypeScript which has recently gained a lot of popularity. Created by Microsoft in 2012, TypeScript is a superset of JavaScript, which means that you can still write normal JavaScript code for your web application, however, you can also opt to use additional features, which helps to avoid common pitfalls in JavaScript. One of the main benefits of using TypeScript over JavaScript is its type checking system, which allows for types to be statically checked before the code is transpiled to JavaScript. This helps to avoid errors such as referencing a null object by accident, or attempting to access a field that does not exist, because the programmer may not know the exact structure of the data. With TypeScript, one can write 'interfaces' for an object, which defines what fields should be available, and if the programmer does something that may cause an error down the line, the static analyzer can warn the programmer. We decided that we like these extra features that TypeScript has over JavaScript, so we went for it.

Alternatively, we could have chosen to write our code in a language that compiles to WebAssembly, like Rust. WebAssembly is a binary instruction format for a stack based virtual machine that runs in the browser. Rust is a modern low-level systems programming language with manual memory management, similar to C++. One of its main draws is its notorious compiler, which can guarantee memory safety using its infamous borrow checker which validates references to objects in memory and spits out errors, frustrating programmers everywhere when they write unsafe code. It manages to guarantee memory safety, while being on par with C++ in terms of performance. Similar programs written in C++ or Rust usually trade blows in terms of performance. Rust can be compiled to WebAssembly, so it would be totally feasible to write our game in that language. This would also bring some of the same benefits that a typed language like TypeScript has like static type checking, while offering other benefits like better performance. This language would be an excellent choice if we needed to squeeze our every bit of performance possible for our application. Luckily, drawing paddles, balls, and handling collisions is not that demanding for modern computers these days and we can afford a performance hit and instead use a language with garbage collection like JavaScript/TypeScript to make our jobs a bit easier. The borrow checker is wonderful, however, given the limited time for working on this project, we decided against it because it is easier to get something running and working when you don't have to worry about manual memory management.

Clearly, at this point, TypeScript seems to be in the winning spot for our language of choice, at least for the code running in the browser. Next, we needed to make a decision about how the game state would be synchronized between different players. We could either go with a peer-to-peer design where all of the code is client-side, or we could go with a client-server model. Most online multiplayer games I have played personally have used this client-server model (Among Us, Team Fortress 2, Rocket League, <Insert Latest Battle Royale Game Here>...). Also, I have found from personal experience that peer-to-peer multiplayer games tend to be a bit more finnicky. For example, games like Super Mario Smash Bros. Ultimate and Mass Effect 3 Multiplayer tend to have issues when a peer loses their connection, and some players can get an unfair advantage over others if they are acting as the host, since they get the most up-to-date information.

We chose to go with a client server model over a peer-to-peer model, partially because of these past experiences as a consumer of multiplayer video games. We also thought it would be easier to go with a client-server model over deciding some consensus algorithm for the peer-to-peer method. The server is the authoritative source. This also allows us to keep track of user statistics in a centralized location, which we need to satisfy our functional requirements in the Software Requirements Specification. In order to provide a leaderboard feature, we need to store the ranking of each user in a central location, so the decision to go with a client-server model for the game was obvious.

Now that we decided to go with a client-server model, we needed to decide what language to write the backend in. From previous experience, I have found that one of the most frustrating things as either a frontend or backend developer is when the frontend developer and the backend developer have different ideas of what the data looks like, or the data models are out-of-sync. The frontend might be expecting for something that the backend does not provide, while the backend developer might not know what the frontend developer wants. I have found in previous projects that when the backend and frontend are written in the same language, I usually don't have to worry about this happening, because the frontend and backend can depend on the same data models. When the types or interfaces for a piece of data is updated for the backend for example, the frontend also gets that update immediately since it also imports the same dependency. When something breaks because of the data model, the frontend and backend developers notice immediately, and the problem tends to get solved quicker, instead of one party saying something to the effect of "it's a backend/frontend problem". The obvious choice for the backend code then, is also TypeScript.

At this point, it was clear that TypeScript is our language of choice – for both the backend and the frontend code. It was time to choose what frameworks to use. For the frontend, we decided to go with Svelte. The obvious choice would be to go with React. It's the most popular JavaScript frontend framework today, and for good reason. I also have previous experience working with React. However, that would be boring. I convinced my team to use Svelte (mostly because I wanted to try something new). It's been under my radar for a while, and it looks cool. Instead of having a virtual DOM like React, the code is compiled to vanilla JavaScript, HTML, and CSS. Getting rid of the virtual DOM helps with performance: `https://svelte.dev/blog/virtual-dom-is-pure-overhead` However, while this is cool from a technical standpoint, it does not really mean much for us. For this project, the difference would not be noticeable. I also thought that Svelte would be easier for the rest of my team to learn, since its syntax is similar to vanilla JavaScript, HTML, and CSS, while React has some quirks that might contribute to a steeper learning curve like JSX syntax, and its lifecycle methods. JSX is HTML-like syntax in JavaScript code, which can be confusing since it is not the same as HTML. From the examples I have seen, Svelte code was simpler. Also, it was trivial to copy the initial mock-ups that Micheal had made in vanilla HTML and CSS, paste it in Svelte, make some minor tweaks, and build on top of it. I know for a fact that if we had chosen React, that process would have been a bit more involved.

The next choice was to decide what technologies the backend would use. The obvious choice here would be Node.js with the express framework. It's an industry standard, and works extremely well. However, I wanted to experiment and play with a newer runtime: Deno. Deno is written by the same guy who made Node.js: Ryan Dahl. He announced it in 2018 during a talk where he explained some of the things he regretted about Node.js. Deno has some security features that come out of the box, unlike Node.js. For example, it does not have network access, environment variable access, or access to the filesysttm by default. These features are opt-in only with a command-line flag. In contrast with Node.js, these permissions are implicitly allowed. Overall, we had no good reason to use Deno over a more established and popular runtime like Node.js, but it did not matter too much for what we were doing. The functionality we needed was there, and there was a routing library like express for Deno called oak. Plus, Deno has a really cute dinosaur logo:

With the frontend and backend language and frameworks chosen, we now had to decide how the client and server would communicate. When a game is active, we need a connection with low overhead and low latency. It would also ideally be bi-directional and the server or client would be able to send a message whenever it wants to, while also being ready to receive a message at any time. Because we chose to make our project run in the browser, our choices basically boiled down to using either WebSockets or WebRTC. WebRTC is usually used for peer-to-peer applications like video calling, however, it also supports the client-server model. It uses the UDP protocol which has the benefit of really low latency, however, because it uses UDP, the connection is not reliable. Messages can be lost, and we would need to account for that in our code. WebSockets on the other hand, use the TCP protocol, and are much easier to use and understand. TCP makes sure that each message is sent and received using acknowledgements. It also ensures that the messages are received in the same order that they are sent, which helps keeps things simple. The WebSocket protocol is also reasonably fast, and we found it much easier to use than WebRTC.

With the major architectural decisions made, it was time to decide what features we would add to our game. What would differentiate us from similar games? We got a hint of this in the section above, talking about the existing solutions. First, we wanted to give the player a sense of progression, should they choose to log in to our system. They can earn experience points, and unlock fancy new skins. Skins do not affect the functionality of the game, but it is well-known that people love to customize things in games with skins in games like Counter Strike: Global Offensive, Fortnite, and Rocket League. For example, at the time of writing, this shiny knife skin for Counter Strike can be bought for $420.69 USD:

We decided against implementing any sort of monetization for our game. For one, dealing with payment information sounds like a nightmare. Also, we thought it would feel more rewarding if users actually earned the skins that they get to use by playing the game and earning experience points.

We also added a leaderboard feature so that users can compete amongst themselves, reaching for the highest score by playing the most games. Users should also be able to see statistics like their win/loss ratio and total games played. We did not find an alternative game that already has this feature. Powerups were another thing we wanted to add to differentiate ourselves from the classic Pong experience. Surely, there are other games out there with paddles, balls, and power ups, but in my quick searching, of the ones that were online and working, none had power ups. This was a similar story for the ability to play with more than four other players online.

For the login functionality, we initially were going to do an e-mail based magic link signin, like Slack does. We instead opted to go for an OAuth authentication flow because it is more secure than sending an authentication token over email. We chose not to store user passwords so that we do not need to have any sensitive information in our database. Instead, we use a JSON web token (JWT) issued by a trusted party (the OAuth provider), and use a shared secret to verify the token and the data in its payload. We decided to use Auth0 as our OAuth provider because it was the first OAuth provider we found on Google, and it seems to have a generous free tier.

For our database, it really did not matter what we picked. We just needed a simple key-value store to record things like the player's username, their experience points, and their currently selected skin. We went with MongoDB since it is a popular key-value database. We had to make sure that a user could not set their skin to a colour that was not unlocked yet, so we check that a user has sufficient experience points before doing the transaction. We also ensure that no user can set someone else's skin by modifying their own request. This is done by verifying the JSON web token issued to the user, which is also passed to the server. That way, the only skin that a player can change is their own.

For hosting the frontend, we went with the easiest to use service that allows one to host static files. Svelte allows the project to be minified into a bundle of optimized code, which is then uploaded to a server where the main `index.html` file is served. We happened to use Cloudflare Pages, but we could have easily also used something like GitHub Pages, Netlify, or Vercel which all offer the same functionality. Because we are just serving the files statically, and we chose to use client-side routing (as opposed to server-side), we opted to use hash based routing. This means that instead of going to a path like `/lobby` on the website, you would instead go to `/#/lobby`. This has the advantage of not requiring server-side logic for routing.

For hosting the backend, we chose to use Cybera. Cybera is a non-profit organization in Alberta which provides free computing resources to students and entrepreneurs. Alternatives include services like Google Cloud Platform, Linode, DigitalOcean, Vultr, etc. We went with Cybera, because it is free for us to use and experiment with as students. We chose to package the backend in a Docker container to make deployment to Cybera easy. Deno is not in the Ubuntu software repositories by default, so instead of getting our backend to work with a specific setup, we made a Dockerfile so that the deployment of the backend is easily reproducible.

4 Potential Impact on Society and the Environment

To be honest, I don't see our project significantly changing someone else's life for better or worse. Unless this game goes viral, the societal and environmental impact will both be relatively low. It's a simple, fun game that friends could get together and play for a bit. The application is not particularly demanding and we found that the CPU usage for machines running this application does not increase significantly while the game is running. Therefore, the amount of energy used by players is pretty low.

Hosting our project online as a website accessible at polypong.ca undoubtedly has an environmental impact. Servers must be kept online, so that the application is available for anyone to access. Our frontend is hosted using Cloudflare Pages, a service offered for free by Cloudflare. Our domain is also registered with Cloudflare, and we are using their DNS services. Because this is an on-demand service, this means that the servers used to run our frontend code does not always need to be active, if users are not using the website. These resources can be used by other users of the Cloudflare Pages service when our demand is low. Furthermore, Cloudflare appears to be a company which is conscious about their impact on the environment. For example, in 2019, they purchased Renewable Energy Certificates to match their electricity use for all of their data centres and offices around the world: `https://blog.cloudflare.com/the-climate-and-cloudflare/`

Our backend code is hosted on Cybera, a local nonprofit organization in Alberta. Unfortunately, due to how we designed the project, the backend server must run constantly to be ready for a client to connect to it. Also, this code does not automatically scale back when the demand is low. Fortunately, it is not running on dedicated resources, as the CPU and memory resources are shared, so other users of the service can use the resources when we are not.

In my searching, I did not find any information about efforts Cybera is making to lessen their impact on the environment, however, we did find that Cybera is using data science to support green tech solutions: `https://www.cybera.ca/cybera-uses-data-science-to-support-green-tech-solutions/`

5 My Role

For this project, I found myself acting as a senior developer of sorts. I got to make decisions about which technologies we were using, and I found my teammates asking for advice on best practices and such, because I have previous experience with Javascript frameworks like React and Node.js. My team trusted the decisions I was making. Most of the time, when we wanted to start working on a new feature, I would be the one to make some boilerplate code that we would then be able to build on top of. This also had the benefit of showing others examples of how to write code for the frameworks we were using.

I worked mainly on the backend and database side of things. I also touched a fair bit of the frontend code, mainly hooking up core functionality and making sure that the frontend can communicate properly with the server. Basically, I touched almost every element of the project that was non visual. Things like fetching data from the server and displaying them, minus the looking pretty part. I also worked entirely on the authentication system. This involved managing the dashboard on Auth0, and writing code to do the login flow on the frontend. On the backend side of things, this involved code like requiring certain client requests to be authenticated and verifying the JSON web token provided in the request. I also worked on getting our project deployed so that we can play the game on polypong.ca. This involved domain registration, administering DNS records, deploying the frontend to Cloudflare Pages, writing Dockerfiles, and deploying the backend to Cybera. For the database side of things, I wrote the queries and helper functions which would be used in other parts of the application. I also worked on designing most of the tests. The database and server tests

Micheal mainly focused on the frontend, UI design, and making features which I added look pretty. He also had to work a bit in the backend when he wanted to add features to the frontend. As a result, Micheal has a really good understanding of how the frontend works, and a solid understanding of how the backend works. Eventually, we also had to refactor the code and we pair programmed together, fixing bugs, adding features, and fixing more bugs. He came up with the initial UI prototype for the frontend, a large portion of which we have tweaked and kept in the final product. He also worked on getting the game to render on the HTML canvas, and did some geometry math to figure out how to render an n-sided polygon on the canvas, rotate the canvas, draw shapes, etc.

Micheal also figured out how to do collision detection, input handling, and designing the game loop. I pair programmed with Michael a lot over the course of the project, getting things between the frontend and backend in sync, and making sure they communicate with each other. Together, we got a basic game working, one without synchronization. “Basic game” meaning that we got multiple players to connect to the server, get their paddles to move on each others’ screens, and a ball moving. However, at that point, there was no synchronization, and although we could see the ball moving in the same direction on all screens, the movement was jittery because of the lack of synchronization. Michael also implemented a majority of the eleven powerups in the game. I helped to implement some when we were pair programming as well.

Joshua mainly worked on the synchronization algorithm. The algorithm implemented was the dead reckoning algorithm. Basically, each client predicts what will happen in the game, and small corrections are made, which helps make the game appear smoother. There was also one day where we pair programmed together for a bit, and another few days leading up to the project demo date where we programmed together, along with Micheal to get the remaining power ups implemented. Joshua also wrote the initial code we used for the ’Add Ball’ powerup. His main contribution was implementing the Dead Reckoning Algorithm, which makes the clients predict the movement of the paddles so that the game appears to be more smooth and less jittery. This involved simulating the paddle movement for opponents, and also predicting the movement of the ball.

6 History of Changes

```
* 5c58b4e 2021-04-15 Arunscape oh man this test plan is gonna take a while to do
* 115824d 2021-04-15 Arunscape final report: alright I guess I just need to come up with a summary of defects
* 456e8eb 2021-04-15 Arunscape I should really sleep at some point tonight
* 3126c3c 2021-04-14 Arunscape dang that was a lengthy section
* c6ee920 2021-04-14 Arunscape final report: wow this section's gonna be long
* a0d4f00 2021-04-14 Arunscape final report: existing solutions
* cf94d0d 2021-04-14 Arunscape final report: problem domain
* a1dfbdc 2021-04-13 mantifao Adding async so that await works, back to 60 FPS after demo
* b58afcc 2021-04-13 Arunscape remove these buttons bc they won't be implemented
*   fb3fd32 2021-04-13 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 75115ec 2021-04-13 Joshua Chang add logo and favicon
* | d9a020c 2021-04-13 mantifao Switch to 30 FPS for the demo
|/
* 6fe8715 2021-04-13 mantifao Fix bug where Lobby not created when websocket not ready
* 8254b39 2021-04-13 mantifao Fix bug of username being empty: Await getUsername since it is async
* 98b4e09 2021-04-13 mantifao Update local leaderboard for newly created users
* 19eb609 2021-04-13 mantifao Username exists message
* c943793 2021-04-13 mantifao Username exists message
* ef5991e 2021-04-13 mantifao Getting username exists message working
* adf6b9d 2021-04-13 mantifao Display username exists to user if username exists
* 8104398 2021-04-13 Arunscape pin oak dependency so that the server can actually build
*   524b0dc 2021-04-13 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 65431a6 2021-04-12 Arunscape change git log command in report
* | 76fd5cc 2021-04-13 mantifao Fix bug with Arun where game was restarting when a player who had lost closes
* | 706caee 2021-04-13 mantifao Fix not logged in when joining Lobby using a link, add sound effects for xp ea
```

```
|/
* 85a494c 2021-04-12 mantifao Rename bomb - commit 2/2. Should now appear in the powerups selection screen
* 277950d 2021-04-12 mantifao Rename bomb - commit 1/2
* 5972aad 2021-04-12 mantifao Add images for shaped paddles
* db7877b 2021-04-11 Arunscape local leaderboard returns indices :)
* 4f343f8 2021-04-11 mantifao Fix bug causing shrunk paddles to become super expanded when setTimeout expired
*   3f52c62 2021-04-11 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * d0cd47f 2021-04-11 Arunscape fix local leaderboard bug SPOILER: I wasn't sorting lol
* | 1ecf59f 2021-04-11 mantifao pathTrace powerup now working with multiple balls
|/
* a41647a 2021-04-11 mantifao Adjust path trace to work with multiple balls
* 1da988a 2021-04-11 mantifao Another ball small bug fixes to make consistent across all clients
*   4b9ea1f 2021-04-11 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 17d3007 2021-04-11 Arunscape nice we are testing everything on the frontend except for the game itself
| * 9e49e1e 2021-04-11 Arunscape here's a login test
* | 15accea 2021-04-11 mantifao Collision detection now working for different shaped paddles
|/
* 84f1669 2021-04-11 mantifao Fix bug where alert was coming up when lobby_id was ""
*   f93d8f9 2021-04-11 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| *   53d7e6a 2021-04-11 Phlana Merge pull request #19 from PolyPong/add-ball
| |\
| | *   632209f 2021-04-11 Joshua Chang add ball powerup finished
| | |\
| | |/
| |/|
| | * a545f53 2021-04-11 Joshua Chang additional ball
| | * c0c9fa6 2021-04-11 Joshua Chang array of balls working
* | | c9e64f5 2021-04-11 mantifao Update getPaddleY() for 8-12 players
|/ /
* | 32d989b 2021-04-11 mantifao Shapes are now rendered consistently across rounds
* | 760da7b 2021-04-11 mantifao Fix bug where powerup names were not correctly displayed on client (were being
* | 75c96a8 2021-04-11 mantifao CurvedInwards, CurvedOutwards and Bumpy render correctly for one round; fixed
|/
* 2dfb8d9 2021-04-11 mantifao Skins should now be correctly rendered
* cd2b13f 2021-04-11 mantifao Fixing skins
* 000e87c 2021-04-11 mantifao Fixing bug where skins are not showing
* a7b9922 2021-04-11 mantifao Update mergeState
* 502b95b 2021-04-10 mantifao Reduce volume of client-server keypress communication
* ea03666 2021-04-10 mantifao Remove some logs, add some logs
* ad870ba 2021-04-10 Joshua Chang added debugging statments
* 80cd847 2021-04-10 mantifao Better Lobby Names; sound effects that only work in certain browsers (Firefox wo
*   afdc17a 2021-04-10 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 59953fa 2021-04-10 Arunscape here's how we do testing for oak
* | bcfa680 2021-04-10 mantifao Powerups now working properly across clients
|/
* 57446c5 2021-04-10 mantifao Small code refactoring, no additional functionality, just cleaner code
*   500411e 2021-04-10 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 4c7ccd4 2021-04-10 Arunscape test script for server
| * 1e94b08 2021-04-10 Arunscape fix database tests
```

```
* | 0eafcf4 2021-04-10 mantifao Stats are updated and saved in the DB; queried using API and displayed to the
|/
* 17d7cf7 2021-04-10 mantifao This should fix one player game starting
* 048fa33 2021-04-10 mantifao Fix lobby doesn’t exist error
* 1a1467e 2021-04-10 mantifao When a user exits a game, it restarts without them; minor changes so in-game pow
* 2a4c65e 2021-04-09 mantifao Combine Stats.svelte with Leaderboard.svelte, Remove Login and Stats routes sinc
* a5860b6 2021-04-09 mantifao Update Settings.Svelte, tell the user what their current paddle color is
* b557edd 2021-04-09 mantifao Leaderboard almost done, just need place returned for local leaderboard
* a155f05 2021-04-09 mantifao Fixed canvas equal to null bug by clearing all the intervals and remove the even
* c3eb94b 2021-04-09 mantifao Fixes bug where keyDownHandlers were not removed when a game was won (they were
* 543859b 2021-04-09 mantifao Fixed a bug with lobby system: when two players are in a lobby and one lost conn
* bdf06cc 2021-04-09 mantifao Lobby improvements, on closing a window, users are now removed from a lobby
* c40b848 2021-04-09 mantifao Update settings and skin selection pages
* dc9dfec 2021-04-09 mantifao Lobby redesign, works with link sharing and shows when users (registered or gues
* 7ba7b18 2021-04-07 mantifao Fixes a bug where skins were set correctly on the server, but not on the client
* 4c6b4c5 2021-04-07 mantifao Remove lobby_id resetting
* 0a19ca8 2021-04-07 mantifao XP bug should now be fixed
* 8fc7df1 2021-04-07 mantifao Fix bug where XP was not assigned due to lobby_id being reset on the client befo
*   bad880b 2021-04-07 mantifao Merge branch ’master’ of https://github.com/PolyPong/PolyPong
|\
| * c187963 2021-04-06 Joshua Chang replace fewer players when merging states
| * b15fddc 2021-04-06 Arunscape we weren’t specifying the port!!!
| * 3c71308 2021-04-06 Joshua Chang add and remove event listeners when needed
* | ac4755d 2021-04-07 mantifao Fixed game breaking bug where second round of game couldn’t start after first
| | *   1b69116 2021-04-06 Arunscape WIP on master: 9f305e3 we weren’t specifying the port!!!
| | |\
| | | * 82d832f 2021-04-06 Arunscape index on master: 9f305e3 we weren’t specifying the port!!!
| | |/
| | * 9f305e3 2021-04-06 Arunscape we weren't specifying the port!!!
| |/
| * 1f029a4 2021-04-06 Joshua Chang good paddle movement
|/
* d79b9bd 2021-04-06 mantifao Fixed lobby selection bug
* fd04f02 2021-04-06 mantifao Add package.json
* 61bace9 2021-04-06 mantifao Skins are now working! Strong start on lobby selection, need to fix bug where tw
* 64e089c 2021-04-05 mantifao Skins are now working!
*   5717d7f 2021-04-05 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * b6c363b 2021-04-05 Arunscape final report draft
| * c368319 2021-04-05 Arunscape bind:this for canvas
* | 33f7a48 2021-04-05 mantifao Game XP is now added and displayed to the user
|/
* c35b6f8 2021-04-05 mantifao Code cleanup, adding comments
* cb34633 2021-04-05 mantifao Update getPaddleY() for up to 7 players (need to finish 8-12 later)
* 55f9aa7 2021-04-05 mantifao Add attribution
* 708e3ee 2021-04-05 mantifao Path Trace powerup done, better initial ball speed
* 9849a65 2021-04-04 mantifao Distracting background working, stops when player who applied it is eliminated (
* d8bed85 2021-04-04 Arunscape this was confusing the compiler I guess
*   42495dc 2021-04-04 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 97409a6 2021-04-04 Arunscape one space
| * 1866e33 2021-04-04 Arunscape cleanup
| * d831fe9 2021-04-04 Arunscape formatting
* | e3b97e1 2021-04-04 mantifao 2 Player wall collision detection working
```

```
|/
*   2635ddd 2021-04-04 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * e5a8ccb 2021-04-03 Arunscape signup works now
| * 7c088c8 2021-04-03 Arunscape oops
| * 6e5ccce 2021-04-03 Arunscape oops
| * ac7d5c2 2021-04-03 Arunscape signup theoretically works
| * 475f2fe 2021-04-03 Arunscape missing }
| * a0a7a2c 2021-04-03 Arunscape redirect user if logged in but account does not exist
| * 81cc3fd 2021-04-03 Arunscape handle loginwithredirect callback
* | 6074ad7 2021-04-04 mantifao 5 Powerups implemented (bigger, smaller and 3 invisibles)! Need testing to ens
|/
*   69fe1c4 2021-04-03 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 2b9e162 2021-04-03 Arunscape copy, share link works now
| * 344cc95 2021-04-03 Arunscape oops
| * a94864f 2021-04-03 Arunscape create get username endpoint
* | 53f0a36 2021-04-03 mantifao Added list of powerups on each client
|/
* 70804d4 2021-04-02 mantifao Update images, Lobby now waits for everybody and has power up selection
* 081f246 2021-04-02 Arunscape global and local leaderboard now using rest methods, just need to get username
* e8e23a8 2021-04-02 Arunscape setskin works from the frontend now (in the sense that it saves the new value i
* 1b40a09 2021-04-02 Arunscape oops
* ad0b811 2021-04-02 Arunscape we're using rest to handle this stateless data instead of passing it over the w
* c11e716 2021-04-02 Arunscape set skin rest endpoint
* 72a2fb8 2021-04-02 Arunscape remove unused code in server.ts
* 7306fc0 2021-04-01 Arunscape local leaderboard rest endpoint
* 6009ccc 2021-04-01 Arunscape global leaderboard rest endpoint
* ade41a6 2021-04-01 Arunscape getavailableskins rest endpoint
* 759dd24 2021-04-01 Arunscape use auth0client.isauthenticated
*   8f9836d 2021-04-01 Joshua Chang Merge branch 'master' of https://github.com/ECE493Capstone/project
|\
| * bb8bda8 2021-04-01 Arunscape login with redirect also, getTokenSilently() returns a proper token now
| * 9463ba2 2021-04-01 Arunscape implement getxp endpoint using traditional rest for simplicity
* | 5e54f66 2021-04-01 Joshua Chang smoother paddle movement
|/
| * 4821c69 2021-04-01 mantifao Work in progress to synchronize emails and usernames across client and server
|/
* a956424 2021-03-31 mantifao Game background colour is now the same as rest of background
* 5628aca 2021-03-31 mantifao Fix background color
*   ab60467 2021-03-31 mantifao Merge branch 'master' of https://github.com/PolyPong/PolyPong
|\
| * 5c0a06c 2021-03-30 Arunscape leaderboard functionality
| * 4c4bf61 2021-03-30 Arunscape leaderboard also theoretically works now
| * 0e8cf83 2021-03-30 Arunscape add logic for getting skins and setting it (with authentication!) (untested
| * c65fecf 2021-03-30 Arunscape fix some linting and allow port to be specified as an environment variable
| * 26fff32 2021-03-30 Arunscape add leaderboard functions for dbhelper, and tests for all of dbhelper
| * 6c3827e 2021-03-30 Arunscape server can run locally again
| * d6916f4 2021-03-29 Arunscape let's see if this is faster
| * 14b0ff6 2021-03-29 Arunscape oops
| * 3812267 2021-03-29 Arunscape ooh let's try this
| * 3177e7a 2021-03-29 Arunscape pls
| * bd9b095 2021-03-29 Arunscape ohhh it might be a CORS thing
* | 0a6155f 2021-03-31 mantifao When we finish one game, a new game is started!
```

```
|/
* a7d7ffa 2021-03-29 mantifao Improve the animation
* 9c827df 2021-03-29 mantifao Update getPaddleY() to work better, add Game Over text animation
* 1ec3a21 2021-03-29 Arunscape try this port on cloudflare
* 09dd68a 2021-03-29 Arunscape let’s see if this works
* 32ff7d0 2021-03-29 Arunscape try binding to 0.0.0.0
* dec0d48 2021-03-29 Arunscape oops I spoke too soon
* 272adda 2021-03-29 Arunscape aaand theoretically we should be live
* ec610f9 2021-03-29 Arunscape it works
* e8d3dc4 2021-03-29 Arunscape rename it so it works on cybera (because I’m already running another instance o
* 9ccb72f 2021-03-29 Arunscape this should work
* 54788c8 2021-03-29 Arunscape create dockerfile
*   a852a09 2021-03-28 mantifao Merge branch ’master’ of https://github.com/PolyPong/PolyPong
|\
| * 2917325 2021-03-28 Arunscape oops
| * acd4dc4 2021-03-28 Arunscape logic for levelling up and setting skins and getting available skins
| * fa4b545 2021-03-28 Arunscape add some pretty colours
* | cfae28a 2021-03-28 mantifao Gameplay working and smooth, graphicaly glitches ironed out, Client can now de
|/
*   a25d53e 2021-03-28 mantifao Merge branch ’master’ of https://github.com/PolyPong/PolyPong
|\
| * 260a729 2021-03-28 Arunscape undo rotation
* | 4bf30cc 2021-03-28 mantifao Collision detection working across multiple clients, graphical glitches abound
|/
*   6940d71 2021-03-28 mantifao Merge branch ’master’ of https://github.com/PolyPong/PolyPong
|\
| * 42f197a 2021-03-28 Arunscape don’t send websocket buffer
```

```
* | 74d6856 2021-03-28 mantifao Working on getting ball consistent
|/
* 1d95da8 2021-03-28 mantifao Collision detection is poor but working
* 00757ec 2021-03-28 Arunscape send unrotated dy dx for ball
* 8f4370a 2021-03-28 Arunscape skip sending websocket buffer in the update lol
* 035bb48 2021-03-28 Arunscape red paddle should move now on other clients
* 8e626d9 2021-03-28 mantifao Collision detection now works client-side! Ball is also drawn correctly (but no
* 5cf8b12 2021-03-27 Arunscape idk :/
* ac0a03b 2021-03-27 Arunscape oops
* 0e12240 2021-03-27 Arunscape ball should be in sync now
* b8ae6e0 2021-03-27 Arunscape ball moves in sync until there is a collision also collisions are broken
* 3814bc4 2021-03-27 Arunscape refactor
* 61f7b36 2021-03-27 Arunscape refactored game object
* dfee8ff 2021-03-26 mantifao Include interface from PPC (removes error message)
* 3e67b26 2021-03-26 mantifao Login now checks db, we let users set their username on first login
* 0dc17aa 2021-03-25 Arunscape we can see other players' paddles moving now!!!
* a3da283 2021-03-22 mantifao Log In on Home page now prints user email to console if authenticated
* 07f5092 2021-03-22 mantifao Fix Auth0 bugs
* 9c3dd00 2021-03-22 mantifao Update db.ts to import from PP-Common
* aa6ce39 2021-03-22 mantifao auth0Client now in store.ts
* 4d4227e 2021-03-19 mantifao Change default port to 3000
* ec001d6 2021-03-19 Arunscape attempt to start game
* a395ced 2021-03-19 Arunscape moved client side lobby websocket stuff to global scope in store.ts
* ccdf142 2021-03-16 Arunscape why did I change these types lol
* 370549f 2021-03-12 Arunscape delete old stuff
* 895b5d6 2021-03-12 Arunscape format tsconfig
* 7a7f748 2021-03-12 Arunscape idk
* 05fdd37 2021-03-12 Arunscape Revert "install in preinstall?" mistakes were made...
* fbd103e 2021-03-12 Arunscape install in preinstall?
* b22b713 2021-03-12 Arunscape frontend can build now
* 603ebdd 2021-03-12 Arunscape if I have to do another UML diagram I'll wanna fucking kms
* a0ca534 2021-03-12 Arunscape render diagrams
* 65c19d3 2021-03-12 Arun Woosaree Create fr26.puml
* 1b90018 2021-03-12 Arun Woosaree Create fr25.puml
* 0ce9d9f 2021-03-12 Arun Woosaree Update fr16.puml
* 82f5fab 2021-03-12 Arun Woosaree Create fr24.puml
* 3dcd30e 2021-03-12 Arun Woosaree Create fr23.puml
* a16f102 2021-03-12 Arun Woosaree Create fr22.puml
* 74b20bb 2021-03-12 Arun Woosaree Create fr21.puml
* 95b4032 2021-03-12 Arun Woosaree Create fr20.puml
* b47b915 2021-03-12 Arun Woosaree Create fr19.puml
* 38a674c 2021-03-12 Arun Woosaree Create fr18.puml
* fed2552 2021-03-12 Arun Woosaree Create fr17.puml
* 3402647 2021-03-12 Arun Woosaree Create fr16.puml
* 747b23f 2021-03-12 Arun Woosaree Create fr15.puml
* 6b0ac6b 2021-03-12 Arun Woosaree Create fr14.puml
* 1b2907e 2021-03-12 Arun Woosaree Create fr13.puml
* 61dd277 2021-03-12 Arun Woosaree Create fr12.puml
* aa1e2b8 2021-03-12 Arunscape we have a basic database
* 254af7d 2021-03-12 Arun Woosaree Update fr27.puml
* 2649182 2021-03-12 Arun Woosaree Update fr28.puml
* e18eca7 2021-03-12 Arunscape changes
* d339d94 2021-03-12 Arunscape delete fr12 for now
* 86248f7 2021-03-12 Arunscape oops forgot to commit this one
* 5ba66dd 2021-03-12 Arunscape add docker-compose
* 1f4c0b0 2021-03-12 Arun Woosaree hopefully last time updating fr1sequence
* 758aaa0 2021-03-12 Arun Woosaree Update fr1.puml
* f780ed8 2021-03-12 Arun Woosaree Update fr6.puml
* 53a72bb 2021-03-12 Arunscape final fr28 hopefully
* db93bfa 2021-03-12 Arunscape final fr27 hopefully
* 56449d8 2021-03-12 Arunscape final fr28 hopefully
* 92291c5 2021-03-11 Arunscape final fr10 hopefully
* 69de270 2021-03-11 Arunscape final fr9 hopefully
* 36c7fa2 2021-03-11 Arunscape final fr8 hopefully
* 96d460d 2021-03-11 Arunscape final fr7 hopefully
* b056895 2021-03-11 Arunscape final fr6 hopefully
* 8caa1aa 2021-03-11 Arunscape final fr5 hopefully
* ab61aa3 2021-03-11 Arunscape final fr4 hopefully
* 818bbb0 2021-03-11 Arunscape final fr3 hopefully
* 7e35a5f 2021-03-11 Arunscape final fr2 hopefully
* 1d17042 2021-03-10 Arunscape script to generate power up sequence diagrams when we figure out how they’re su
* 91b3235 2021-03-10 Arunscape tweaks
* 0875ad3 2021-03-10 Arunscape remove need for jwt to view leaderboard
* c6a266b 2021-03-10 Arunscape play game does not involve lobby
* e0930b3 2021-03-10 Arunscape check for 3+ users fr3
* cbeddbd 2021-03-10 Arunscape fr12sequence
* c21c408 2021-03-10 Arunscape fr28sequence
* 7fb4f70 2021-03-10 Arunscape fr27sequence
* 2879334 2021-03-10 Arunscape fr10sequence
* 7360541 2021-03-10 Arunscape fr9sequence
* 4e54651 2021-03-10 Arunscape fr8sequence
* 51c2cc5 2021-03-10 Arunscape fr7sequence
* 72d270e 2021-03-10 Arunscape fr6sequence
* 49b9cd4 2021-03-10 Arunscape fr5sequence
* 9d5c1ef 2021-03-10 Arunscape fr4sequence
* 7aad86c 2021-03-10 Arunscape fr3sequence
* fe90914 2021-03-10 Arunscape fr3sequence
* dd09d87 2021-03-10 Arunscape update fr2sequence
* 33bcadc 2021-03-10 Arunscape update fr1sequence
* 8ac287e 2021-03-10 Arunscape fr2sequence
* ff37e84 2021-03-10 Arunscape fr1sequence
* c6c7837 2021-03-10 Arunscape the websocket is now accessible from anywhere in the application
* 04c6bb7 2021-03-09 Arunscape use the types I defined for transmitting messages
* de4b815 2021-03-09 Arunscape add joinsuccesspayload to payload.ts
* c9a8575 2021-03-09 Arunscape fixed the enter lobby id bug. now uses the textbox input
* e250620 2021-03-08 Arunscape here's what the communication protocol might look like
* f9e9273 2021-03-08 Arunscape hey we can now create a lobby, and join it with a unique id!!!
* b180a2f 2021-03-08 Arunscape add uuid
* 0d70162 2021-03-05 Arunscape package-lock go brrrrrrrrrrrrr
* 95a3d19 2021-03-05 Arunscape document how to install private package
* f2eb902 2021-03-05 Arunscape create polypong-common package
* c6d248a 2021-03-05 Arunscape v0.0.1
* 0a64f51 2021-03-05 Arunscape move Game types to npm package
* 51b2d4e 2021-03-01 mantifao Player number done; paddles rotate accordingly
* 0d60aee 2021-03-01 mantifao Ball now scales based on client's window size
* c5d9a42 2021-03-01 mantifao Added a ball and basic collision detection
* 7cc2cc0 2021-02-28 mantifao Cleaned up code, paddles have colors!
* ffad707 2021-02-28 mantifao Paddles are now bounded!
* 0dfdaf9 2021-02-28 mantifao Bottom paddle moves! No boundary checking yet
* dccca91 2021-02-28 mantifao Initial paddles are drawn
* 0fb8f2c 2021-02-27 mantifao Adding classes but not yet working
* d81ecac 2021-02-27 mantifao Shapes are drawn again
* 6f9fadb 2021-02-25 Arunscape work breakdown structure
* 345723b 2021-02-17 Arunscape idk what I changed but it works beautifully now. If the user is already signed
* e07fb58 2021-02-16 Arunscape getting somewhere
* 34625da 2021-02-16 Arunscape hm
* 61928d8 2021-02-16 Arunscape add auth0 client library again
* d158d9c 2021-02-16 Arunscape oh boy lots of changes
* 5c62216 2021-02-15 Arunscape add auth0 client library
* ccea9cc 2021-02-08 Arunscape frontend mockup but now in Svelte still kind of ugly because we're not reusing
* 739fbe9 2021-02-08 Arunscape actually let's try svelte@next
* f51de3c 2021-02-08 Arunscape add lockfile to gitignore for now
* 3573a86 2021-02-08 Arunscape use typescript
* 0acc958 2021-02-08 Arunscape init svelte
* 4369d8e 2021-02-08 Arunscape move frontend mockup
* bd0191d 2021-02-08 Arunscape copy over stuff from main branch
* 7b293ed 2021-02-06 Arunscape signin works
* d38e4b4 2021-02-06 Arunscape signin works
* 026077b 2021-02-05 mantifao Create README.md
* 61a93f6 2021-02-05 mantifao Delete .DS_Store
* 279df4a 2021-02-05 mantifao Delete .DS_Store
* c3a08ef 2021-02-05 mantifao Delete .DS_Store
* 58569af 2021-02-05 mantifao Delete .DS_Store
* 1d00abd 2021-02-05 mantifao Delete .DS_Store
* 1aab6ff 2021-02-05 mantifao Delete .DS_Store
* fe524da 2021-02-05 mantifao Add gitignore
* 72beaa3 2021-02-05 mantifao Initial mockups
```

7 Summary of Defects

oh boy....