**Sprint Report: Milestone 2A**

The tools we’re using for this project are Google Drive for documents, GitHub, Unity, and Visual Studio for source code, and Discord for communication.

**What we planned to do**

We plan to implement a MySQL database or simple text files to store persistent information. If we decide to use MySQL, it would be best if we could use either php or javascript to communicate between Unity and MySQL. In terms of networking, we discussed how we wanted to implement multiplayer and considered the amount of messages we’d have to handle if we were processing messages from 4 players. We came to the conclusion that a screenshot-streaming technique similar to how Amazon AppStream works would be a simple solution. This was in part because we felt that it would be much simpler than sending multiple transform messages to the server, and the server sending these transform messages to every single client that was connected to the game. Because neither of us have done this before, we plan to research various methods to accomplish this. At the moment it looks like we’ll be using Application.CaptureScreenshot to get the png image, convert that to a byte array and send it across the network. Lastly, we plan to add a type of latency mitigation which is a combination of lockstep and time manipulation. When the server detects discrepancies, it would stop all the clients, process the game states and stream screenshots of the valid game states at a higher fps rate to get to the current state without losing seconds on the timer.

**What we accomplished**

We accomplished establishing the database by using XAMPP’s built in MySQL server managed by the phpmyadmin interface. In order for Unity and MySQL to communicate, we ended up using php and Unity’s WWW Form library. The scripts authenticate the username and password and returns if it’s valid or not. It also checks new accounts, and sees if the username already exists in the database or not. On the networking side of things, we managed to successfully send screenshots to multiple clients over the same network. We decided not to use Application.CaptureScreenshot, but instead used a render texture to decrease the time it took to capture the screenshot and convert it to a byte array. When a client connects, it allows them to see their movements on the screen. In terms of game mechanics, we incorporated a power up that increases the blast radius of the bomb when picked up.

**What we didn’t accomplish**

The screenshot-streaming technique took longer than we anticipated, so ultimately, we couldn’t incorporate latency mitigation into our build. Our database half is also on a separate branch from the networking half and we didn’t have time to merge them. Although we allow multiple clients to connect to the server, it doesn’t actually instantiate more than 1 player yet.

**What we plan to do next**

Login-wise, we plan to display error messages whenever a username doesn’t exist in the database or whenever the passwords don’t match with what was stored in the database. Likewise when creating an account, we also need to display appropriate error messages like the username has already been taken or when the passwords don’t match when validating. We realize that it’s also important to merge our database and networking branch as soon as possible so that’s the first thing on our list. After merging, it’s important that we don’t stream the game yet before a player’s login information has been verified. So we talked about implementing a lobby where the players wait for at most 60 seconds or when 4 players connect before starting a game (or rendering the screenshots). If time permits, we plan to incorporate our latency mitigation idea as well and work towards completing the requirements for Sprint 3.