Bubble Bursters - Documentation and User’s Manual

By Nathan Pham, Jonathan Duong, and Sharanya Sudhakar

Table of Contents:

1. Prerequisites
   1. System Requirements
   2. Required Software
   3. Network Setup
2. Game-Play Instructions
3. Coding Documentation
   1. Registering / Unregistering
   2. Lobby
   3. Creating a Lobby
   4. Joining a Lobby
   5. Leaving a Game
   6. Application Specific Protocol
4. Network Gameflow
5. Packet Structure
6. Self-Grading Report
7. Prerequisites:
8. System Requirements

In order to play the game, both server and client computers must have a Windows 10 Operating System.

1. Required Software

To compile, to run, and to step through our code and inspect every model in detail, Unity Engine 2018.3.2f1 must be installed. Running any other version breaks the game due to mismatch in the libraries.

To open the project, extract the folder to any destination, open Unity, in the top right corner, hit Open and navigate to the extracted folder.

Once the project opens, you can access our project folder -> Assets -> Scenes -> Main\_Menu.unity to run the full game while inspecting the code. The source code can be found under project folder -> Assets -> Scripts.

To build the game, click File -> Build Settings -> Build then choose a directory. After building, open the executable.

However, if you just want to play the game without, follow Section II. Running the executable with “css432project” under the game folder on both computers is enough.

1. Network Setup

From our developing experience, Unity Engine does not work too well with the university’s private network due to the firewalls and administrative permissions to allow TCP connection to run in the school’s Windows Lab computers with Unity installed, and for devices not being able to discover each other. In other words, this game **will not work on the school’s network** for reasons discussed later in section 3A. Even from home, it can be a hassle if the network itself is not properly initialized at both ends. Out of the following, please check that each of these solutions have been used together before claiming it is our network code’s fault.

* Try to ping the client and host computers. If this fails, check that your firewall is disabled before server client connection, and retry the server and client. If this fails, enable currently connected network on both ends as private networks and repeat again.
* Run telnet on both computers to see if the port 2485 is enabled. In Windows, the command “netstat -ant” (without quotations), will show listening and established connections on both ports. If telnet runs fine, proceed with the game’s server and client again. (Nota Bene: telnet clients and server may need to be installed in your computer if you do not already have them)
* Set up TCP rules in all computers. Create TCP rules by going through Control Panel -> Security -> Firewall -> Advanced Settings -> right clicking on both Inbound and Outbound connections, and allow port 2485 in TCP to happen in private networks. Rerun the tests above, and retry connecting the game’s server and client.
* Alternative: Set-Up an Ad-Hoc Network. Due to restrictions in Windows 10, it’s no longer possible to create an adhoc network via GUI. The only possible method is through command line. First, open a command line prompt with administrator permissions. Type in the following line of code to see if creating an adhoc network on your wifi card:  
     
   **netsh wlan show drivers**If the line “Hosted network supported” says “Yes”, then proceed. If it does not, then it is not possible to create an adhoc network<> Then type in the following command.  
    
  ***netsh wlan set hostednetwork mode=allow ssid= key=***   
    
  Type in any name you want for SSID and any password after key. For example, if the SSID is set to AdhocTest with a password of 12345678, other devices will see AdhocTest as a separate network and connect with the password 12345678. **Note: The password must be 8 characters or longer**. After an adhoc network has been created, start the network by typing in the following command:  
    
   ***netsh wlan start hostednetwork***Doing this will start the adhoc network and allow other devices to connect. Later on if you want to stop the network, type in the follow  
    
   ***netsh wlan stop hostednetwork***These instructions were adapted from this [website](https://www.howto-connect.com/create-wireless-ad-hoc-internet-connection-on-windows-10/): <https://www.howto-connect.com/create-wireless-ad-hoc-internet-connection-on-windows-10/>

II. Game-Play Instructions

Our game has been uploaded to a github repository, and access is granted via a link (<https://github.com/Tofumars/CSS432Download>). This link provides a github file which contains a zip file of the game’s executable. Please note that the code itself is under the bitbucket project shared with the instructor.

STEPS:

1. Download zip file from github link: <https://github.com/Tofumars/CSS432Download>
2. Unzip and extract the gamebuild.
3. Connect to the network (Have both test devices set to using private network)
   1. If you are in your house, Using your Wifi and setting it to a private network is enough.
   2. If you are in the university network, set-Up an Ad-Hoc network as described in I.C.4. Set this as a private network on all testing devices.
4. Run the game, and allow the program to communicate over private/public networks.
5. Press play and choose a name. The name box **cannot** be left blank.
6. Click the toggle to become a client, otherwise continue
7. Click “Find a Lobby” / “Create Lobby”

SERVER:

1. Please wait about 5 seconds after the client presses join in order to start the game

CLIENT:

1. On the left side, there will be a panel with servers. When the server shows up, click the join button
2. Wait for the server to hit start game

III. Coding Documentation

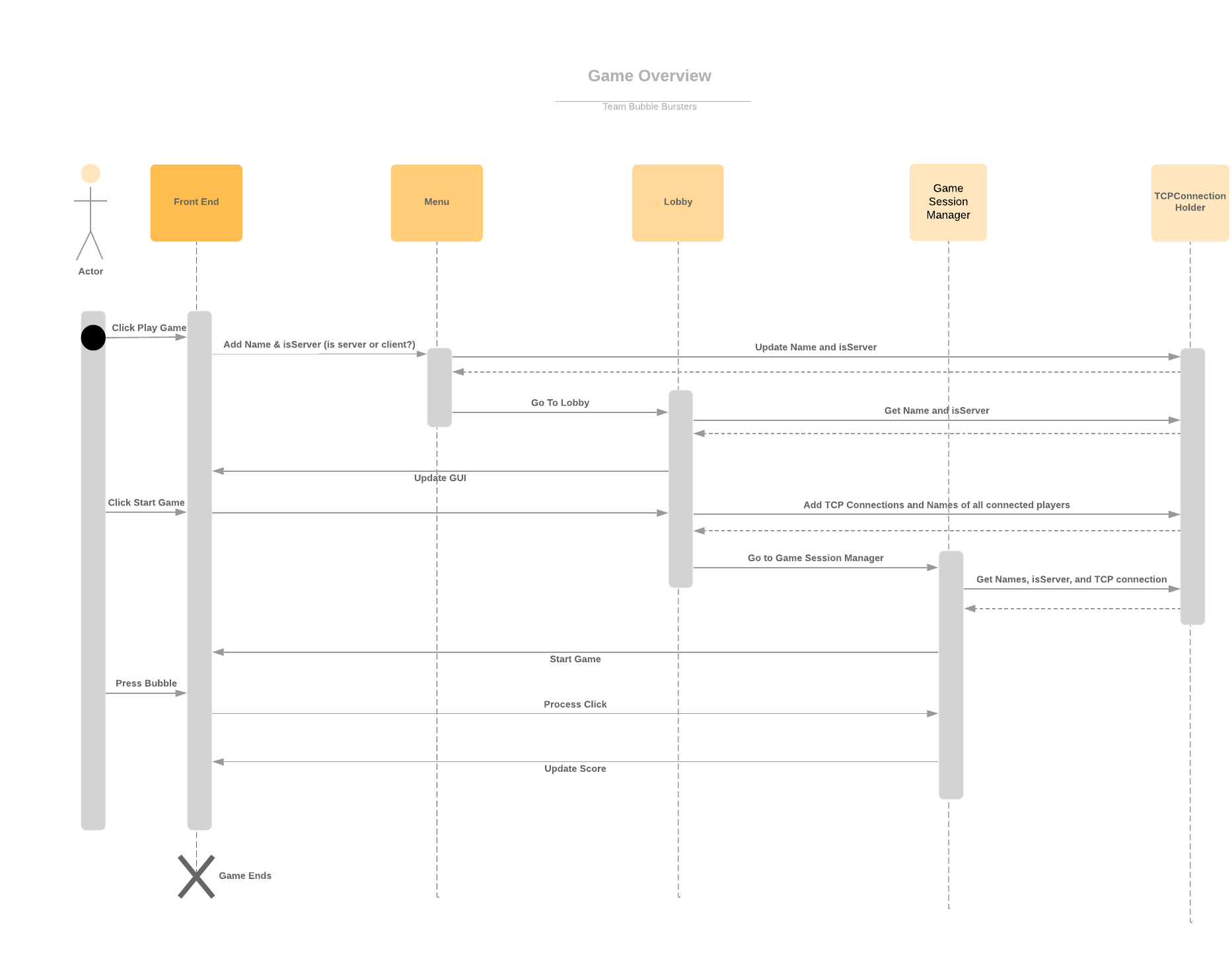


Figure: - High Level Overview of the process

Bubble Bursters uses both UDP and TCP protocols to send data. The classes MyUDP and MyTCP are wrappers for the socket. UDP protocol is used to broadcast and to initialize a TCP connection whereas TCP protocol is used to send data about the state of the game (i.e bubble clicks, bubble positions, scores, etc).

**Menu** - Start menu where user can input their name and decide to be a client or a server

**Lobby** - Menu that displays the list of current lobbies if a client, otherwise the menu hosts the server

**Game Session Manager -** The Game logic that spawns the bubbles and synchronizes the server and client games.

**TCPConnectionHolder** - An object that is passed between Menu, Lobby, and Game Session Manager. The TCPConnectionHolder contains the following information:

* Name of player + other players
* isServer - whether or not the player is a client or a server
* TCP connection - connection between the client and server. Passed between the Lobby and GameSessionManager.

1. Registering / Unregistering

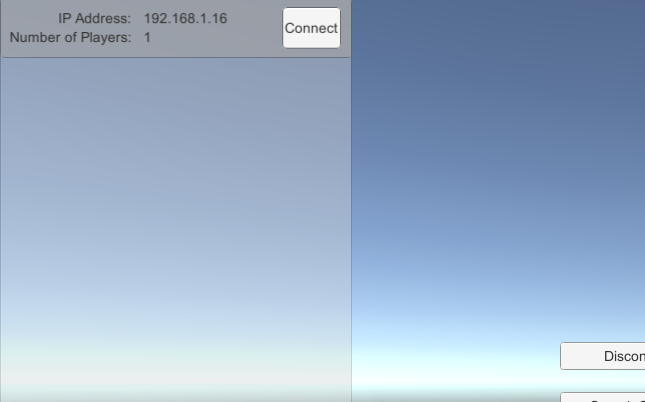
While searching for players, the Server constantly broadcasts “BLAB” packets over UDP. The BLAB packets contain the following information:

* Server Marker (indicates it is a server)
* BLAB Marker (indicates that the server is open)
* Server IP (for identification purposes)
* Number of Players currently in the server

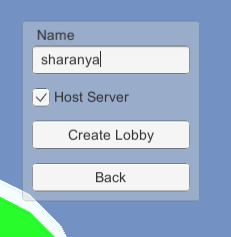
This is most likely the reason that our game does not work over the UW Network, as the network likely not only blocks these kinds of broadcasts, but also may route our connections to different routers in the network.

When a server receives a request to join from the client (as mentioned later in Section 3D), the server records the client’s name, IP, and time joined in a dictionary. This dictionary is used to identify who would like to join. Clients cannot join if their name is not in this dictionary (i.e, if the client tampered with a packet to skip the “BLAB” phase). If there is no response of any sort from the client after 10 seconds, then this entry is removed from the dictionary.

1. Listing Lobbies



The Client listens on the network for any open lobbies, this happens immediately after the client registers. The server starts sending BLAB packets to registered clients. If the Client receives a “BLAB” packet from a Server, the Client lists that Server in the Server Panel on the right side, along with the IP and the number of currently active players. All currently active servers sending BLAB packets get listed in the Server Panel.

1. Creating a Lobby

On the main menu, there is a toggle to choose whether to be a Server or a Client. If you are the Server, the lobby only has a start game button, that lets you begin the game, when other clients have joined. On the client side the lobbies currently active are listed if the player joins as a client with the toggle of. At this point, the client has to wait for the lobby server/host to start the game. Once the game starts the lobby closes, as no new player can join in the middle of a game.

1. Joining a Lobby

As mentioned in the “Listing Lobbies” section, if the Client receives a “BLAB” packet from a Server, that Server’s game will be listed in the Server Panel. On that entry is a connect button. When pressed, the Client sends a “BLAB” packet itself, containing the following information:

* Client Marker (indicate that it’s from a client)
* BLAB Marker (indicate that the client wants to join)
* Client IP (to be able to resend)
* Client Name ((for identification purposes)

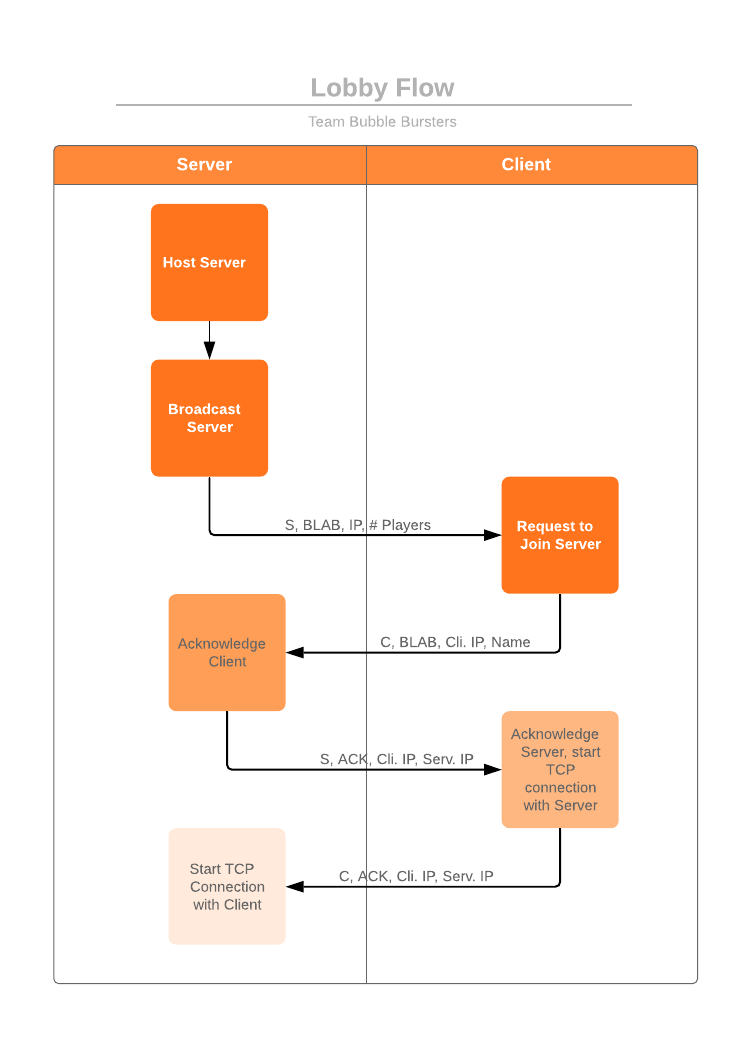
This BLAB packet is sent to the Server over UDP. When the Server receives this BLAB packet, it indicates to the Server that a Client wants to join. The Server then sends an ACK back to that Client containing the following information:

* Server Marker (to indicate it’s from a server)
* ACK Marker (to let the client know that it’s received its message)
* Client IP (to let the client know it’s to the right client)
* Server IP (to let the client know which server is responding)

This ACK packet is sent to the Client over UDP. When the Client receives this ACK, it creates an ACK of its own containing the following information:

* Client Marker (to indicate it’s from a client)
* ACK Marker (to let the server know it’s ready to join)
* Client IP (to let the server verify which client is joining)
* Server IP (to let the server verify that the client is joining the right server)

After receiving this ACK, the Server uses the information exchanged from all the ACKs and BLABs to start a TCP connection with the client. This completes the 3-way handshake.



1. Leaving the Game

To leave the game, you can press the “Quit” button in the main menu. Furthermore, when the client joins a server, the client can press the “Disconnect” button to disconnect from a server and close their window to leave the game. As long as the client is not connected, the client can leave at any point in the lobby scene.

In addition, during a game session, the client can close their window and the game does not crash.

1. Application Specific Protocol
   1. Network Gameflow

The transport layer protocol used in the application protocol is TCP, and it is handled by the MyTCP class, which is a higher-level abstraction designed and implemented by Jonathan and Nathan. Using the MyTCP class as a basis, the ServerManager and the Client Manager classes were implemented.

The MyTCP class takes advantage of callback functions to allow receiving and connecting sockets to be asynchronous to improve responsiveness.

Figure 1 illustrates interaction between the server an 2 clients. The ServerManager sends a gamestate packet to all connected clients every 6 frames, giving them information of which bubbles are popped, and updated player scores. Every time the client clicks, the client sends a packet containing the click location of the mouse, and the server interprets whether a bubble is at that area, and will either grant them points or not.

* 1. Packet Structure

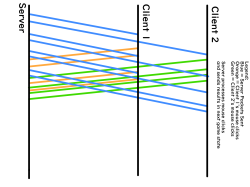
Figures 2 & 3 below illustrate packet structures used by the server and client, respectively. Figure 4 illustrates the circle packet structure.

When any of these packets are sent, they retrieve the data from the current gamestate and convert the data into an array of bytes. These bytes are the payload of the sent TCP packet. When the receiver receives the TCP packets, the bytes are deserialized and converted to their proper representation.

If the receiver is a client, the Client Manager will queue the server packets, and the bubble spawner extracts the packets to spawn the circles, and the scores are updated.

If the receiver is a server, the Server Manager queues the client packets, and the game session manager extracts the packets and raycasts to check if a bubble is popped at the clicked mouse location.

**Figure 1: ServerManager-ClientManager Communication**

****

**Legend:**

**Blue - Server Sending Packets,**

**Orange - Client 1 sending mouse clicks**

**Green - Client 2 sending mouse clicks**

**Figure 2: Server Packet Structure**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **float** | **int** | **Circle Packets** | **string** | **int** | **string** | **int** | **string** | **int** | **string** | **int** |
| Time Remaining | Number of Circles (N) | N Circle Packets | Player 1 Name | Player 1 Score | Player 2 Name | Player 2 Score | Player 3 Name | Player 3 Score | Player 4 Name | Player 4 Score |

**Figure 3: Client Packet Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **string** | **float** | **float** | **float** |
| Player Name | Mouse x | Mouse y | Mouse z |

**Figure 4: Circle Packet Structure**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **char** | **float** | **float** | **float** | **float** | **bool** |
| Circle Type | Current Life | xpos | ypos | zpos | isPopped |

IV. Self-Grading Report

Nathan Pham contributed to the application-layer protocol of the game, and created the scoreboard and the basis of the MyTCP class. In addition, he designed the architecture of the UDP communication used to discover player and servers in the lobby network, the packet structure, and the concept of the game. Furthermore, Nathan developed the entire game logic such as the GamesessionManager, the BubbleSpawner, and the Circle classes. Nathan wrote Section I, Section III - subsection F, and the Self-Grading Report.

Jonathan Duong contributed to the implementation of the lobby, connect/disconnect, and the register/unregister portions of the game. He also developed the main menu as well altering the Lobby and GameSessionManager to support transferring information in-between scenes. In addition, he helped Nathan restructure the MyTCP class with asynchronization, and both Jonathan and Nathan worked very closely with each other in the last 3 weeks of development. This explains why most of the development during this time was pushed by Jonathan (Nathan was having a difficult time pushing into BitBucket via their GUI interface SourceTree, and kept crashing in his Microsoft Surface). During this time, the Lobby and the UDP protocol used for communication were refined, and peer programming was implemented between Jonathan and Nathan to discover bugs and to have a better understanding of WireShark. Jonathan guided Sharanya in the documentation in Section III, and wrote the instructions of how to set-up the Ad-Hoc Network. In addition, he created the flow diagrams for the lobby connections and the overall program flow.

Sharanya Sudhakar’s work was to implement the game’s UI, and the scoreboard and timer, as well as the join/exit and register/unregister functions. However, she did not complete the required tasks in the game’s proposal. Jonathan Duong ended up completing the join/exit, and the register/unregister functions, and Nathan Pham had to restructure her scoreboard and timer classes completely to make it fit into the networking code. Sharanya attempted to give Nathan and Jonathan a more aesthetically pleasing UI, but Nathan declined the push, stating that pushing a new build the night before the presentation would be a bad idea due to the probability of unforeseen bugs. After the presentation, Sharanya is left to write the documentation in section III, subsections A-E with the guidance of Jonathan.

V. Extra: Video recording

<https://photos.app.goo.gl/e9qbhEWFKmg4xJM28>