1. **Network Messages and Protocols(All messages sent have <EOF> attached by default)**

Login Protocol

|  |  |
| --- | --- |
| Client Messages | Server Message/Response |
| *“Attempting Login: (username)|(password)“*  -client sends its username and password to gain  access to server | *“Login Success (username)”*  -Server sends success message and  username if the username and password  match the database entry  *“Login Failed”*  -if username and password does not exist or  not matching |

Register Protocol

|  |  |
| --- | --- |
| Client Messages | Server Message/Response |
| *“Registering: (username)|(password)”*  -client is attempting to make a new account for the  game. | *“Registered (username)”*  -server sends back this message if successful along  with the username  *“Invalid Registry”*  -if the proposed username and password do not  uphold the database requirements |

Disconnect Protocol(Client to Server)

|  |  |
| --- | --- |
| Client Messages | Server Message/Response |
| *“Disconnect Me (username)”*  -client tells the server it wants to disconnect and will promptly shut down its socket connection | nothing is sent back, the server will remove the client from the server and from the game if they are in one |

Game Select Scene Protocols

|  |  |
| --- | --- |
| Client Messages | Server Message/Response |
| *“Find game (lobbyname)”*  -client asks the server if the specified lobby exists  already | *“Game Found”*  -server found the specified game and tells the client to either join or otherwise  *“Game DNE”*  -server could not find the specified lobby  *“Game FULL”*  -server found the game but it has the maximum  number of player |
| *“New Game (lobbyname)”*  -client is attempting to make a new lobby instance | *“Game Created”*  -server makes the new game and moves the client into it  *“Game exists already”*  -attempt to make new lobby fails because it exists |
| *“Joining (lobbyname)”*  -client is attempting to join the lobby it found | *“Join Approved”*  -server says it’s perfectly valid to join the existing  game |

Lobby Protocols

|  |  |
| --- | --- |
| Client Messages | Server Messages/Response |
| *“Awaiting Game (username)|(lobbyname)”*  -client tells the server to hand over the list of users in the lobby at this instance | example response:  *“P1L: (username)|(x)|(z)”*  -server tells the client all users that are in the  lobby, number in the message after the “P”  refers to the index in the list. username  determines what the name is for the index |
| *“This player is ready (index)|(lobbyname)”*  -client lets the server know it’s ready to play | example response:  *“P1R: ready”*  -server let’s all other clients know that this client  is ready to play |
| *“This player is not ready (index)|(lobbyname)”*  -client tells server to tell the others | example response:  *“P1R: not ready”*  -server let’s the others know this client isn’t  ready to go |
| *“Remove me (index)”*  -client tells the server to take it out of the lobby | *“Lobby Refresh”*  -server tells whoever else is left in the lobby to  prep for new list of users |

Game Messages

|  |  |
| --- | --- |
| Client Messages | Server Messages/Response |
| *“Player;(index);(alive?);(x);(z);(xv);(zv);(lobby);end”*  -message is sent to server to echo to all players in  the update  -index refers to players index in the server’s game  -alive? refers to the player’s status in the game,  either dead or alive  -x, z, refer to positions and xv,zv, refer to the  velocities | *“Player;0;T;9;8;0end;1;T;8;9;1end;2;T;9;7;2end;3;F;9;6;3end;”*  -similar to client’s but it sends ALL players info to ALL clients playing |
| *“Bomb;(x);(z);(strength);(lobby);”*  -x,z pos of the bomb, strength of the bomb refers to how far the explosion will go | *echoes the message to ALL clients*  *this way bombs are regulated through the server* |

1. **Text Description of every Script/Class written for this project**
2. “bombHandle.cs”
   * manages any bombs that are backlogged in a client received by the server
3. “bombLogic.cs”
   * specifically used for when a bomb is placed and explodes, it will create the explosion collider
4. “chat.cs”
   * handles lobby chat messages and displays it in the specified UI object in the lobby scene
5. “Client.cs”
   * contains the: StateObject class and Client Class
   * script handles all data sending and receiving to and from the server and responds to most of server messages in order to get the game client running
6. “esplode.cs”
   * handles the explosion effect on screen, mostly dealing with when it needs to disappear
7. “frequency.cs”
   * handles the powerup for increasing the number of bombs the client has.
8. “Game.cs”
   * handles all game specific data and data only for the client to read off of during a game instance
9. “gameselect.cs”
   * deals directly with the specific UI logic in the Game Select scene
   * works to display the popup windows and querying the server for games
10. “impact.cs”
    * deals directly with the collision detection with bombs and setting up the explosion effect
11. “lobby.cs”
    * deals directly with the UI logic in the Lobby scene only
    * handles the ready up and the list of names in the lobby
12. “login.cs”
    * handles sending the login information to the client, where the client will relay that to the server
13. “MD5Manager.cs”
    * built specifically for hashing/encrypting strings for the passwords
14. “Parser.cs”
    * parses all messages received by the client
15. “playerMovement.cs”
    * handles all user input to the game object to move and drop bombs
    * here is also where the client will send its position and bomb drops to the server
16. “register.cs”
    * deals directly with the register page of the game client
    * it will check the info submitted to see if it’s valid and will send when it’s valid
17. “scoring.cs”
    * script deals with the scoreboard component during game runtime
    * will display all users in the lobby and their status of alive or dead
18. “Settings.cs”
    * contains the class “IPManager” where it contains the usual IP addresses of the team and will be used to check which one needs to be connected to
19. “UImethods.cs”
    * deals with all usual methods of transitioning from one scene to another for button presses on each scene and is used to obscure the passwords on the UI elements
20. “Server.cs”
    * Main server code and logic that handles all message processing, connection listening, and basic game overview and lobbying.
21. **System Requirements**

* Unity 5 for clients
* Windows 7
* Visual Studio 2010 (minimum) for the server
* MySQL server ver5.6 (recommended to have the workbench to check the database) for database

**Tutorial Section**

1. Setting up the database to accept logins, you have two options:

*\*\*Regardless of the options below, you need to have the references attached in your version of the solution:*

* *System.Data*
* *MySql.Data*

*\*\*You must also have the C# MySQL connector, which if you need to download and install the link is here:*

* *https://dev.mysql.com/downloads/connector/net/6.9.html*

a. Pseudo-Database interactions

-In the "ServerSystem" Folder open the ClientServer solution and open up the Server.cs file.

-There is a class called "Settings" in the near top of the code and there is only one variable, database

-switch to false if you want a Pseudo-Database intereaction or True if you wish to use a functioning

MySQL Database

b. fully functioning MySQL database

-Make sure the database bool variable in the server.cs file is set to true for this to work

-Setup your MySQL database to have the schema named "bmdb" and the table "main"

-Schema needs to have the following columns and types

1. username - VARCHAR(12)//the number doesn't entirely matter, cannot be null

2. pass - VARCHAR(12)//cannot be null

3. wins - INT

4. games - INT

-enter the ip address of your database in the server.cs file in either under the "IP" class in the string

"mySQL" OR directly in the "DatabaseHandler" class under the server variable

After following the steps above, you should have a database or pseudo-database interactions with the game

2. Setting up a game with two players:

-open the unity project and select the scene "LogInScreen" and run it

-If you are using your own MySQL database, proceed to the register page to create logins and follow the

onscreen instructions

-Build and run to get a second game client running to test this

-If you create a login successfully, you will be immediately transported to the Game Select scene

-Whenever login is successful you will be taken to the Game Select

-Create a Game will bring up a popup with an input field

-enter a name and you will be brought to a lobby

-The game will not start if there is not at least two players ready'd up

-Second client should use the search feature to find the game you made and if found, it will bring up a popup

for you to join it

-When both players are ready, the game will start.

\*\*If a player disconnects, the game treats it as a death. So with only two players it will result in a game over

and bring the remaining player back to the lobby\*\*

3. Running two or more Sessions at once

-Build and run at minimum of three client while having the unity editor open and running

\*\*currently our game doesn't really account for having one login logged in at one given instance, basically the

same login can be used over and over\*\*

-Have two clients create a game session and name it separately from each other because it will make sure

that all games are unique.

-Have the other two join each one, 1 for 1 and the other for the other.

-once ready'd up a game will start.

-Games are separate and run on their own.