Project Initial Design

Team Name: the-a-team-but-not-in-the-ed-sheeran-way
Group: Chris Hinstorff
Derek Benson

We have decided to adapt our project to focus more on the real-time communication between players that are connected to the system. Players can maintain a list of friends and have 1-on-1 conversations with these friends. They can also create and join chat rooms to communicate with people. Results of the game are announced to all the friends of the participants that are online.

The main challenge of this project is writing the concurrent chat server. We considered two possible choices for the design of this component:

- This first was a multithreaded python server that communicated with clients over a socket connection. We decided against this model because we would have ended up implementing a number of things that erlangs distributed communicated model gives us for free. That lead us to the second option.

- Erlang gives us a great communication model for message based concurrency between clients and different servers. The clients will rely on making RPC calls to the Erlang Chat server for things like sending messages or managing/viewing their friends list. Erlang game servers will respond to requests to create a game lobby and then execute moves by the players of those games.

We will additionally have a GUI component (chat box, etc) that the users can interact with. This will be done in Python. We will use Erlport to communicate between the two components.

We have decided on and implemented the message system in Erlang
messages handled by the login/logout server
{login, Name}
{logout}
{add_friend, Name}
{remove_friend, Name}
{list_friends, Name}

messages handled by the room server

{list_rooms}
{create_room, RoomName}
{create_private_chat, [Users to Add]}

messages handled by the client

{online, [users online]}
{msg, Room, Contents}

messages handled by the chat server

{subscribe, Client}
{unsubscribe, Client}
{send_msg, Msg}

Messages handled by the leaderboard server
{add_score, Game, User}
{subscribe, Client}
{unsubscribe, Client}

The examplestate.pdf file shows the basics of having two clients connect to a chat server, send a message to eachother via chat, and then start a game.

Now that the protocol has been implemented we need to run some tests and hook up as basic UI to make sure that things are working.