**Trivia Game Implementation Description**

The trivia games server makes use of both TCP and UDP communication in its implementation. TCP sockets are used exclusively in the user registration server for the initial registration of players. The user registration server keeps a connection active only long enough to complete the request in a manner similar to HTTP requests. A game client attempts to connect to the registration server. Once a connection is made the client sends the registration request. The server then responds and closes the connection. The commands available to the client in this mode are Check Server, Register User, Deregister User, Ready, and Not Ready all in a pull format. Once all users are marked as ready the game server closes the registration and takes control from the user registration server until the active player list is copied to the new game and the old active player list is cleared, at which time registration will be reopened and the user registration server will regain control.

The trivia game server makes use of UDP multicast to communicate with its clients and primarily functions using a push format. The trivia game server operates asynchronously. One thread consistently sends out messages held in its send queue, while the other thread receives data and makes use of the observer pattern to notify games when new data has been received. The server consists of six commands, five are push and one is a pull. The commands that will be broadcasted to clients consist of sending the question, finding who pressed the buzzer, allowing a user to attempt an answer, notifying the user the points they earned with 0 points meaning an incorrect answer, and start/unfreeze screen. The only pull command is the buzzer which is sent from the client as soon as they want to answer. To guarantee that all clients have received information commands are rebroadcast after a set period of time until all users acknowledge they have the data up to a set retry amount.

**Trivia Game Responsibilities**

* Game Server’s Responsibilities
  + Keep track of player scores/metrics
  + Create and maintain trivia games
  + Serve questions to players
  + Select which player can answer
  + Verify if a selected answer is correct
  + Give points based on the remaining time left for the player to answer
  + Freeze and Unfreeze players
* Game Client’s Responsibilities
  + Display questions to the user
  + Gather information on the user for registration
  + Allow the user to register for a game
  + Prevent the user from attempting additional answers if first attempt is incorrect
  + Adjust the amount of time for answering each question between the min and max time limits
    - This is done by keeping track of the number of consecutive right or wrong answers
  + Enforce answer time limit once a player is allowed to answer by the server
    - Example: When a player is allowed to answer by the server they will have 5 seconds to select and submit the right answer. If no answer is selected in that time frame an empty UUID (one with all 0s) will be sent by default to the server. If an answer is selected, but not submitted within the time frame the currently selected answer’s UUID will be sent.
  + Freeze/Unfreeze client input when commanded by the server
  + Notify the server when the player ran out of time to answer
  + Notify the server when a player leaves
  + Display the player’s current score and all other player’s scores

**Server and Client Finite State Machines**

For the most part the Finite State Machines (FSMs) are extremely similar. The server and the game rules, at least flow wise, are almost one to one comparison wise. The only major difference between the FSMs is between the client and the game rules. The difference between those two FSMs is that our current client displays the answers when the question is received, and the buzzer and answer submit are done when the user submits an answer.

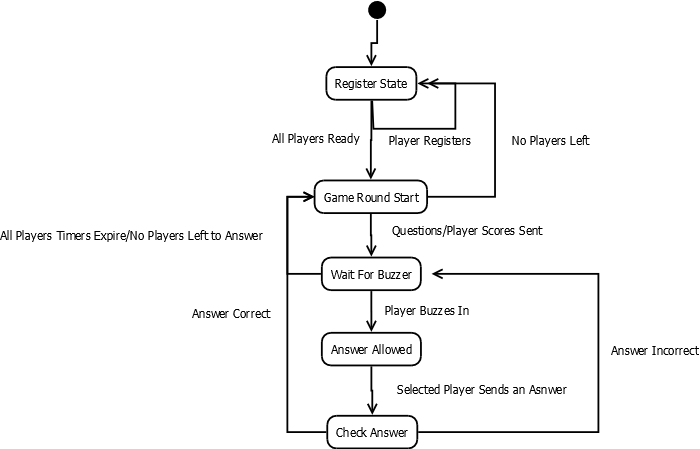


Figure 1. Server Finite State Machine

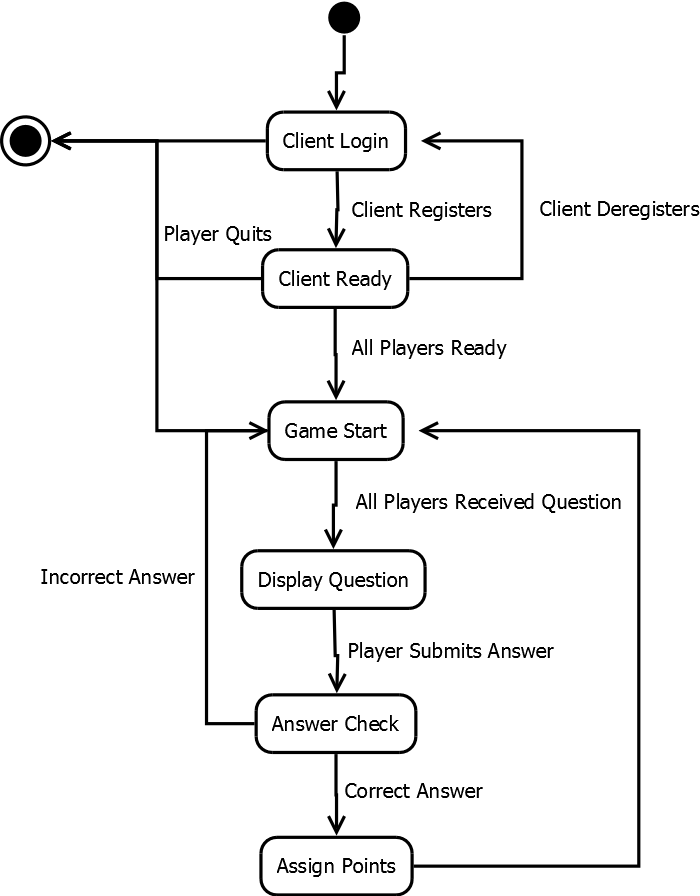


Figure 2. Client Finite State Machine

The advantages and disadvantages of the network protocols that we implemented as discussed above with respect to supporting the game rules follows:

**Further Discussion**

* Advantages
  + TCP registration ensures that all the players that request to play the game will be positively registered.
  + UDP allows for a nearly unlimited number of players and does not significantly increase the bandwidth requirements as the number of players increase.
* Disadvantages
  + Using UDP to the clients may cause a problem in one phase of the game. After all the players respond that they have received the question, the order to display the answers may not make it to one of the participants. This will not be known by the game server.

TCP was the network protocol used in the registration portion of the trivia game. UDP is used to play the trivia. The commands implemented in the game on the sever and clients preserve the game rules even with the contestant workstations being mostly latent. The logic in the game allows each contestant workstation can be taken care of individually so that they are aware of the others score and when another workstation buzzes in but they do not need to be visible and accessible to each other.

Multicasting is used to implement the behaviors and communication tying the game server and contestant workstations together. Multicasting using UDP can function to preserve the game rules even with a large scale number of users. With multicasting, each constant workstation can simultaneously be given messages containing any of the necessary information for gameplay.