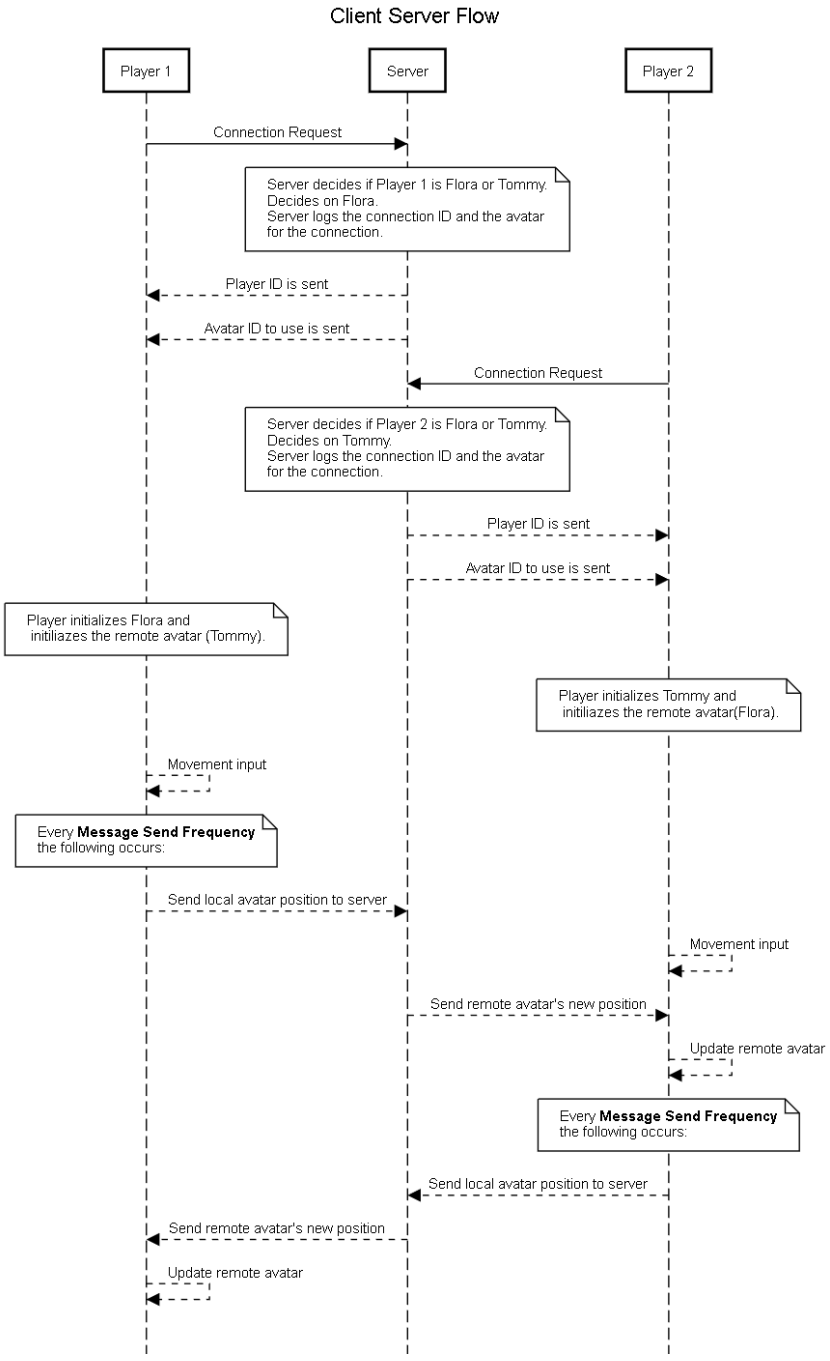
Assignment 4 Report

1. Document the message flow between the client and server, showing the steps from a client requesting a connection, through to the first position update message being sent by the client and that position update being received by another client. Use a UML sequence diagram. (If you need a refresher, look at the example in lecture 03-3.) Provide approximately one paragraph of text with the diagram to explain its contents.



Paragraph for question 1:

The diagram shown previously is the Client Server flow sequence diagram for assignment 4. In this diagram, we can see that when a player starts the client they initiate a connection request with the server. The server uses internal logic to determine whether the client is Flora or Tommy and logs the connection ID. The server then sends a message to the client with the clients' player ID and follows up with a message containing the avatar ID it must use. The client initializes the appropriate avatar, and logs the player ID and avatar ID. It also initializes the remote avatar. From there, the client sends a message every *messageSendFrequency* containing the clients' local avatar position, rotation and movement state. The server receives this message and sends it to all the other clients. These other clients update their remote avatar based on the appropriate algorithm.

2. Explain how you would extend this solution to handle more than two avatars. In what ways does the current solution embed knowledge that exactly two avatars are being used? What specific parts of the code would you need to change?

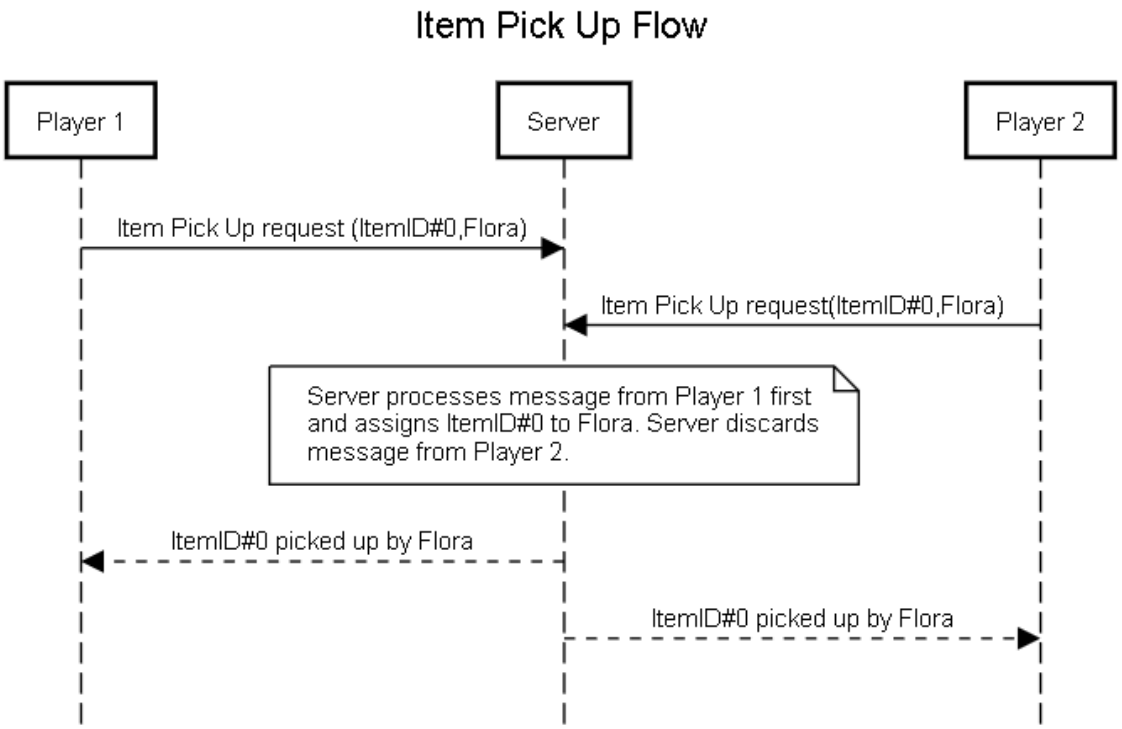
To extend this solution to handle more than two avatars, I would have the server instantiate a new avatar in the game at a location that is empty (that is, devoid of an avatar). The client would then have to instantiate its avatar in the correct location. The server should alert all clients of the newly instantiated avatar so that they may instantiate said avatar at the appropriate location as well. This avatar would be assigned an avatar ID and they would be removed if the player disconnects. It would be logged with the connection ID and the player ID. The current solution embeds the knowledge that exactly two avatars are being used because it chooses from Flora or Tommy. Since there are only two avatars to possess, no more than two clients can be connected. Specifically, the AddNewClient code in the *ServerNetworkScript* should be modified to be able to handle more than two avatars.

3. You have used the simplest form of smooth corrections in this assignment. Describe how this algorithm sometimes leads to unaesthetic character movement. Explain how you might elaborate this system to provide more realistic smooth corrections.

Currently, the smooth correction simply moves towards the target location at double speed and teleports to the target location if it is lower than some threshold distance. To provide a more realistic and aesthetically pleasing smooth correction, I would elaborate the system by having the character properly turn to face towards the correct direction and animate the turning, then proceed to animate the running towards the correct location. This would take longer but would lead to a more realistic smooth correction.

4. Imagine you were to extend this "game". Objects appear on the ground, and the first player to walk over the object collects it. (After an object is collected, it disappears.) What is the primary implementation challenge with this game feature? Briefly sketch how you would use messages to implement this functionality. What new messages would you introduce? (Specify the names and parameters of the new messages, and explain how they are used.) What two quality attributes do you need to trade off in the implementation of this game feature?

If I wanted to extend this "game" to add the ability to pick up an object and collect it, the primary implementation challenge with this feature is the possible game state conflicts that could occur due to different client states. It is possible that client 1 picks up the object before client 2, but due to propagation delay, client 2 is the first to confirm the pick up with the server. I would introduce two new messages named *requestPickUpItem(itemID,avatarID)* and *itemPickedUp(itemID,avatarID)*. This message is sent from the client requesting the pick-up, but the server holds a canonical representation of the world state. If two clients submit a request at the same time, the first message to arrive on the server would be the one that is processed correctly, and all remaining clients are sent the *itemPickedUp* message. The server updates it's canonical record of events (i.e. either player 1 or 2 have the item) and all clients update their world state based on the *itemPickedUp* message. If two messages are sent at the precise same time, whichever message is processed first is the one that is interpreted as correct. Any *requestPickUpItem* message with an already picked up item ID will be discarded. This would end up increasing animation delay and response time as all clients would not see the result of that action until it has thoroughly been processed by the server. Only when this is done is the original client informed of having properly picked up the item and thus the animation plays.



5. This solution uses two channels: a lossy channel and a lossless channel. Explain the difference between these types of channels (i.e., what do *lossy* and *lossless* mean). Explain what kinds of messages the lossless channel is used for. Explain what kinds of messages the lossy channel is used for. Explain why it is useful to have both types of channel.

The lossy channel is an unreliable channel wherein the message that is sent through it may be lost or dropped due to network conditions, etc. The lossless channel is a reliable channel that ascertains message delivery but may not ascertain the proper order of the messages delivered. The lossless channel is used for any messages that are game-critical and extremely important to game state. For example, the lossless channel is good for game start messages, game win/loss messages, death messages, item trading messages, etc. The lossy channel is used for any data messages that are not game-critical. For example, position updates, server log messages, animation/AI state updates, etc. It is useful to have both types of channel because always using the lossless channel can become quite expensive in terms of network performance. It can cause an unnecessary amount of network latency for messages that are altogether not too important. Particularly, looking at position updates, since they are frequent, losing a message is OK because we will most likely get another shortly thereafter. However, we can not always use the lossy channel as some message are of high importance and occur infrequently. Game win/loss/start messages for example are extremely important and it is OK to incur the network latency cost for such an infrequent, yet important, message. Having both allows us the ability to choose and minimize the network latency cost.