CSc 631/831 HW 4

Unity code should be at Git and reflections are Individual submission.

(Due by 3/30 Mon 11:55 pm @iLearn) Team of 2 or 3.

1. As a team, one person is responsible for a server code and the other for a client. If you make a team of 3, then the 3rd person helps on both client and server part. Please specify your role when you are submitting your reflection.

2. You develop your part using the given code set. You may choose to use a different server code to start. If so, you need to provide detailed description on the server code and protocol, so I can get clear idea that you studied the server code and not simply utilize the blackbox tool. You need to teach your part to your members and you need to learn other parts from your members. **Write a reflection** on what you taught and what you learn. You learn a lot while you teach! You need to document meeting time and the location and the people in the meeting.

3. **The server code** is provided to test simple Login protocol. You need to create 4 new protocols that are needed for your game including login and registration if your game needs them. List protocols that will be needed for your game and develop the protocol and support the protocols using the server code and the client code. **Write a reflection** on the classes that you learned and you developed, and short description on them and their relationship (inheritance or composition)

4. **The client code** is provided to test login. You will need to create a new scene for your own game and copy the classes and code to build your own game scene. As there will be 2 teams for each game team, please separate scenes or protocols, so they don't overlap. **Write a reflection** on the classes that you learned and you developed, and short description on them and their relationship (inheritance or composition

5. Integrate your team's code using Git. Unity and Git don't work very well, so you need to practice and figure out your team's rule to be able to integrate effectively. **Write a reflection** on this experience and policy within your team.

6. For AR team, you may choose to work on MagicLeap and its special functionality – opaque rendering and object tracking as HW 4.

7. Submit the document, share your git with me (ilmiyoon@gmail.com) (required) and upload all the codes (zip) to iLearn, if possible.

Client Server Architecture

(For HW 4)

A. Basic Knowledge on Client Server Communication

1. Client and Server communicate by sending and receiving messages. (Request message and Response message)
    a. Client sends request message to server : server receives request message (and process request)
    b. Server sends response message back to client : client receives response message (and process response).

2. Server who receives request message needs to run infinite loop not to miss any requests from clients.
    a. (JAVA server code) uses serversocket to create a server. (Provide a port number)

        serverSocket = new ServerSocket(configuration.getPortNumber());

    b. Within the infinite loop, server is waiting for a connection. Once a connection is established, a socket object is returned. The socket object represents client (remotely). From this socket object, inputstream (read request) and outputstream (send response) are retrieved.

    c. Usually server program uses multithreaded programming to improve performance. In game server, we create a new thread per client and maintains N threads for N connected clients.

3. Difference of Web Server and MMO Game Server
    a. Web server serves multiple clients concurrently using multithread (similar to game server)
    b. Web server doesn't synchronize all connected clients. Game server needs to synch all the clients (for example racing game with 4 players, all 4 players need to be synched)
    c. How does game server synch all clients?
        i. Broadcasting mechanism (server broadcasts and clients are listening). This requires client listen to in-coming message all the time.
        ii. Polling mechanism - client checks server frequently to retrieve any pending updates. WoB game server/client uses this mechanism. Server maintains Update_table listing all clients and client status. Client uses "Heartbeat" protocol (typically 10 times per second) to retrieve this table. – note: this table is in memory.
        iii. Persistent data (DB) needs to be updated frequently (?) not to lose data in case of crash. But DB update doesn't need to be 10 times a second!!

4. Create a new protocol – similar to define a new user-defined function. You need to define unique protocol ID and the sequence of parameters and their types
    a. Example of protocol ID

```
// Request (1xx) + Response (2xx)
    public final static short CMSG_AUTH = 101;
    public final static short SMSG_AUTH = 201;
    public final static short CMSG_HEARTBEAT = 102;
    public final static short SMSG_HEARTBEAT = 202;
```

    b. **In server side**, you need to extend GameRequest & GameResponse objects. For example, Login protocol extends both and RequestLogin object is a sub class of GameRequest. Once you extend the abstract super class, you need to implement "parse" and "doBusiness" methods. Here, for Login protocol, parse reads the userID and passwd from the message and doBusiness is checking with DB for authentication.

    c. When a request arrives, the first 2 bytes (short int) is used to figure out the whole length of the (Request) message. The next 2 bytes (short int) is used to figure out which protocol it is. And using that procotol ID, a proper request object (RequestLogin in case of login) is instantiated.

    d. In "doBusiness", you create responseObjects and insert to list of GameResponses. When the "doBusiness" is complete, it sends out all the responsesObjects to client.

    e. In client side, you do same thing!

5. Understanding Client-Server communication from client side
    a. Client sets up asynchronous communication. It sends request and then moves on instead of waiting to receive response. To be able to support this, "callback" mechanism needs to be set up. For example, before sending "Login" request to server, callback for "Login" response should be set up. Function pointer (Delegate in case of C#) can be nicely used to register a callback method/function for specific response.

B. (Client side) Code and classes that you can reuse
    1. /Assets/Network

    Every cs file in this folder is essential. The code below shows how you can add a new protocol as request message.

```
public class NetworkRequestTable {

public static Dictionary<short, Type> requestTable { get; set; }
```

```
public static void init() {
	requestTable = new Dictionary<short, Type>();
	add(Constants.CMSG_AUTH, "RequestLogin");
	add(Constants.CMSG_HEARTBEAT, "RequestHeartbeat");
	add(Constants.CMSG_PLAYERS, "RequestPlayers");
	add(Constants.CMSG_TEST, "RequestTest");
	}
```

The code below handles the response message wrt its protocol.

```
public class NetworkResponseTable {

	public static Dictionary<short, Type> responseTable { get; set; }

	public static void init() {
		responseTable = new Dictionary<short, Type>();
		add(Constants.SMSG_AUTH, "ResponseLogin");//201
		add(Constants.SMSG_PLAYERS, "ResponsePlayers");//203
		add(Constants.SMSG_TEST, "ResponseTest");//204
	}
```

2. Assets/Scripts

   Constants.cs defines all protocols.

```
// Request (1xx) + Response (2xx)
public static readonly short CMSG_AUTH = 101;
public static readonly short SMSG_AUTH = 201;
public static readonly short CMSG_HEARTBEAT = 102;
public static readonly short SMSG_HEARTBEAT = 202;
public static readonly short CMSG_PLAYERS = 103;
public static readonly short SMSG_PLAYERS = 203;
public static readonly short CMSG_TEST = 104;
public static readonly short SMSG_TEST = 204;
```

   MessageQueue.cs and `ExtendedEventArgs` are essential.

   Main.cs needs to include the code below;

```
	public class Main : MonoBehaviour {

	void Awake() {
		DontDestroyOnLoad(gameObject);

		gameObject.AddComponent<MessageQueue>();
		gameObject.AddComponent<ConnectionManager>();

		NetworkRequestTable.init();
		NetworkResponseTable.init();
		}

	// Use this for initialization
	void Start () {
		SceneManager.LoadScene ("Login");
		ConnectionManager cManager =
		gameObject.GetComponent<ConnectionManager>();

		if (cManager) {
			cManager.setupSocket();
```

```
                StartCoroutine(RequestHeartbeat(1f));
                // in case you need “requestHeartbeat”
            }
        }
```