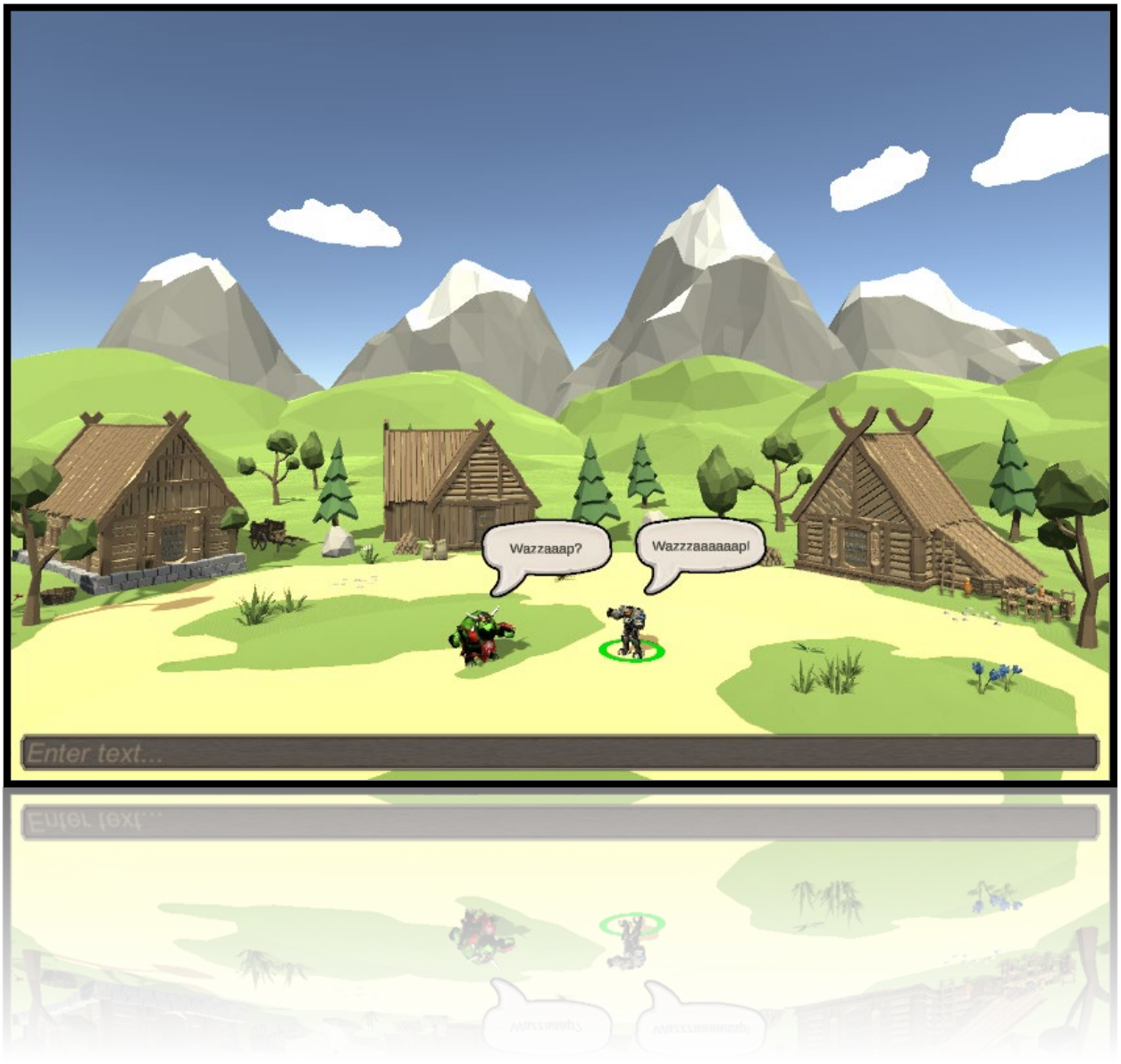
Networking - Assignment 3 - Implementing Mini Town



Getting started

In this third assignment you have to implement a 3D chat lobby client/server setup, *using the starting code* provided in the assignment zip file. The main difference with the second assignment is the *kind* of messages that are being sent between the client and the server and *how* they are being sent; we are no longer using simple strings, but Packets or Objects instead. For this assignment you'll have to implement the server, the ChatLobbyClient class on the client and use/extend the given shared library to provide the requested features.
Before you start on the actual assignment, follow the steps below to get acquainted with the given setup.

Getting started with the Mini town client

1. Open up the Mini Town Unity Client project, open the 'ChatLobbyClientScene' and press Play.

2. You should see a little town with some moving clouds and a sun, but other than that it is still fairly empty.
   In the top left of the UI you see a bunch of buttons to help you fix that, try them out!
   As you can see, all of them do exactly what their title says.

3. Select the TestUI object in the hierarchy (it can be found under Canvas/TestUI):
   a. Read its documentation component & the source code of the AvatarAreaManagerTester.
   b. Check how the buttons on the TestUI are connected to the public methods of AvatarAreaManagerTester.

   This TestUI allows you to test the town's functionality locally without any networking.
   The goal of this assignment is to remove that TestUI and make the town a networked chat lobby where every avatar represents a single client. For example, 'Add random avatar' will be replaced/triggered by an actual client joining, 'Speak MORTAL!' will be replaced by an actual client talking, etc. (See below).

4. Inspect the ChatLobbyClient GameObject/Script & the AvatarAreaManager GameObject/Script.
   Both GameObjects have some notes added and both Scripts are documented.

   The most important thing to understand for now, is that the *ChatLobbyClient* (the script you will have to modify for this assignment) has a reference to an *AvatarAreaManager*, which you can use to add avatars, (re)move avatars, get an avatar reference to let it speak etc & that *how* you actually do that is demonstrated in the methods of the given AvatarAreaManagerTester (and also in the ChatLobbyClient.showMessage method).

   For this assignment, most of your work will take place in the ChatLobbyClient Script and the shared package.
   You should *not* have to change any code in either the AvatarAreaManager or the AvatarAreaManagerTester (unless you are going to try and do really fancy stuff, yes you know who you are).

5. As you can see in the Project Window, this project has a bunch of assets, but you *don't* have to touch them.
   The whole project is set up in such a way that you can complete the assignment by modifying the ChatLobbyClient script (and disabling the TestUI). If you *are* interested in each and every file/folder, check out the *readme* in the *Assets* folder root & inspect the notes on each GameObject in the Hierarchy.

Getting started with the Mini town server

1. Open up the server project and look at the project layout:
   a. there is one server project and one shared project
   b. the server project is defined as a Console Application in its Properties (a .exe)
   c. the shared project is defined as a Class Library in its Properties (a .dll)
   d. the shared project is referenced by the server project and automatically build when the server is built
   e. the shared project has a build event which copies the latest dll to the shared folder of the client
2. Check out the source files for the shared project. The StreamUtil, Packet and ISerialization classes discussed during lecture are already present, as is an example SimpleMessage in the protocol folder.
3. Check out the TCPServerSample and notice how it is already set up to accept and handle multiple clients.
4. Notice how the server is still an echo server. Whether we send a string, a packet or an object, at this point the server doesn't really care. It just reads any bytes it can get and sends them back to sender.
5. Run the server, start the client in Unity, add some random avatars and type a chat message.
   Try to find out the flow of events that leads to a random avatar showing your message.
   Make sure to ask questions during lab if any part of this is unclear.

Now that you've worked through these steps, you are all set to start working on the actual assignment detailed on the next page. Read through the *whole* assignment, before you start.

Sufficient requirements (Basic avatar chat):

In order to pass the 'Sufficient' for this assignment, implement both the functional and technical requirements below:

Functional:

- The server maintains a list of avatar model objects, one for each accepted client:
  - Each avatar model instance has an id, skin and position values
  - When a client joins, the server registers a new avatar model instance with a unique id, random skin and random but *valid* position
  - When a faulty client is detected, its avatar model is removed from the server administration
  - Whenever the server avatar administration is updated all clients are informed
- Clients visually show all avatar models a server is maintaining (and any changes in a client's avatar model). (using the provided AvatarAreaManager and AvatarView)
- A client can chat 'through' his/her avatar,
  in which case the entered text will show through the chatbubble of the client's avatar on all clients.

Technical:

- Communication is no longer based on string parsing, but Packet or Object based (see lecture 3)
  (also see the 'Very good' requirements, before you decide which one to choose. You can always switch between approaches, but that is of course more work than making an informed decision up front).
- All messages are validated/enforced by the server, the client does **not** pass its own id.
- A single client action causes a single server response. For example, when a client joins, don't send a Packet for each client individually, but combine all required information into 1 Packet/Object.

Good requirements:

In order to pass the 'Good' for this assignment, implement the additional functional requirements below:

Functional:

- A client can move his/her avatar by clicking anywhere in town, the client's position changes on all other clients as well. The server should validate whether the given position is legal.
  (Tip: use _avatarAreaManager.OnAvatarAreaClicked to get the clicked position)
- If you type /whisper in front of your message, only clients within 2 units of your own avatar get to see your text. This is enforced by the server, not by the client. (You can use the last clicked position for all avatars and don't have to take walking into account).

Very good requirements:

In order to pass the 'Very good' for this assignment, implement the additional technical requirements below:

Technical:

- Communication is Object based (see lecture 2)
- Protocol messages correctly serialize their contents (e.g. AvatarModel : ISerializable)

Excellent requirements:

In order to pass the 'Excellent' for this assignment, implement the additional functional requirements below:

Functional:

- There is a ring around your own avatar so that it is clear which avatar is yours
  (tip: the easiest way is to update the avatar view prefab and add a ShowRing method to the AvatarView)
- You can change the skin of your avatar, either through chat or through some other means.
  (tip: use the SetSkin method of the avatar view, the change may be random)

TIP: Testing your setup with multiple clients

For this project it is *no* longer possible to duplicate the "client" within the current project like we did for the chatbox. There are however still several ways to test this setup with multiple clients:

1. You can build the client and start it multiple times.
2. You can clone your project using ParrelSync (https://github.com/VeriorPies/ParrelSync).