|  |  |
| --- | --- |
| **COMP260 – Distributed programming** | **Worksheet 2** |
| **Network programming in Python** | |

**Introduction**

The goal of this worksheet is to experiment with socket-based networking to make your SUD use separate client and server components.

**Exploratory Programming**

Take the Python and C# samples from the lecture and make them run on your computer. You should be able to see that it is possible to mix and match the server and client components (a C# client and a Python server and visa versa). This is possible because sockets define the operation and data interface between the two application roles (client and server). As long as we adhere to those definitions, we can share data with sockets across any platform that implements the socket standard.

**SUD Review**

Did you manage to build a SUD last week? If not, don’t panic, I’ve included my SUD project in learningspace. Have a look at the Python project to see how the classes work. Use the class hierarchy and state diagrams from last week’s session to see how the code solution has evolved from the initial design. It’s worth remembering that technical design is not a straight jacket definition of how code will be implemented, but a guide which is always open to change

**SUD to network SUD Theory**

Once you are comfortable with how your (or my) SUD works, you need to refactor it into client and server projects. The first part of this is to work out what functionality will exist in the client and server components.

I’ve spoken briefly about MVC-based applications (model, view and control components). For the SUD, the dungeon is the model, as it describes the network of rooms the game contains. The view component is the part of the game that prints out responses to what the player does, and the control component is the part of the game that takes user input and updates the model (dungeon) based on what the player wants to do.

Typically, a network-based game will place the model and controller components into the server (to stop people hacking) leaving the client to implement the view and a lightweight controller that will take user input.

**SUD to network SUD Development**

To develop the network SUD, you will need to work what classes go into which applications (client or server). You can do this by making a component diagram which will show the two applications and their class organisation and functionality.

In addition, you need to think about the data communications, what messages will the client send to the server and what will the server return to the client? The simplest approach, is to take whatever the user types in and send that to the server, then process the client command on the server and send the resulting text back to the client.

**Implementation advice**

1. Make two new Python applications, one called client and the other called server.
2. Use the Python client and server examples from the lecture as a starting point for your applications (copy & paste the code in).
3. Slice and dice the existing SUD functionality so that the client is responsible for user input, sending it to the server, receiving the server’s response and displaying it on the screen
4. Slice and dice the existing SUD functionality so that the server is responsible for manging the dungeon, receiving client input, processing it and sending the results back to the client.

**Taking it further**

Once you have a basic dungeon crawler, it’s worth looking at the following:

1. Develop a network noughts and crosses with 2 clients and one server. The server will be responsible for managing the game mode and controlling game states and the clients will take it in turn to move. Given the requirements for the clients to be able to view the board, you will need to think about how you will send the board state to the clients and make sure that the server is responsible for all updating and control of the board, rather than letting the clients do it as this creates a lot of scope for hacking and data mismatch.
2. Develop a chat service, where multiple clients can chat using a shared server. This should create some interesting challenges in managing ad hoc client to server communications.