Project 1 - A Guessing Game

an introduction to network servers

I've chosen a number between 1 and 100. Can you guess it?

The Goal

In this project you develop a TCP/IP based server that implements a number guessing game. This assignment is an introduction to TCP/IP socket programming and connecting clients and servers together.

You will only be responsible for the **server** side of things, you will use the `telnet` program for testing and validation.

The Game

The game is to guess a number between 1 and 100. After a client connects, the server will silently choose a random number and give higher/lower hints to the player as they try to guess the chosen number.

With a Partner, Play a Few Rounds of the Guessing Game

Question

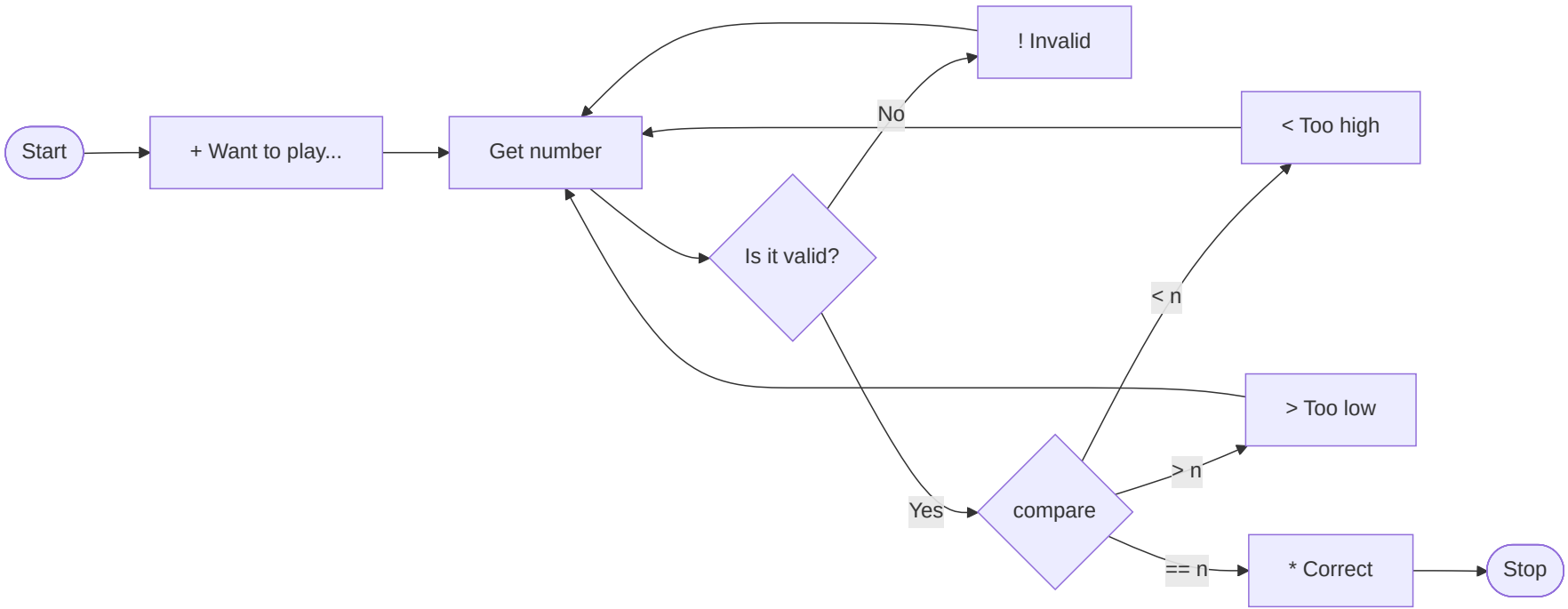
Write down the sequence of play

- How do you start?
- What do you do if they give a bad response?
- How do you end?

07:00

▶ Start

Guessing Game Flow



The Guessing Game Protocol

Guessing Game Server

Requirements

The server must:

- Accept network connections on a configurable port number (>1024)
- Send the `welcome` message at the start of the communication.
- Read a line of data (terminated by newline).
- If the input is not a number or not 1, 100, send the `invalid` message. Otherwise, send either the `higher`, `lower`, or `correct` message.
- Loop until a correct guess or the connection terminates.

Guessing Game Server

Requirements

Non-Requirements

The server must not:

- Accept guesses via the terminal or shell where it was started.
- Display a graphical interface. This is a **server**. Servers run in the background and have **no user interface**.
- Terminate unexpectedly on invalid input or sudden connection loss. It should be reasonably resillient.

All interaction is over the network

- If you need diagnostics, you **may** print them to `stdout` of the server program.

Guessing Game Server

Requirements

Non-Requirements

Your Code

The Programming Part

- Use any programming language you want; Java, Python, C, Go, Rust...
- Use the **socket** library, do not use a higher-level library.
- You should be using functions like `accept()`, `send()`, `receive()`, `read()`, `write()`.
- You only have to handle one connection (client) at a time.
- If you are able to handle multiple clients at once, you will be ready for Project 2. This typically requires multi-threading of some sort.

Guessing Game Server

Requirements

Non-Requirements

Your Code

- **Java**

Java

An excellent starting place for **Java** is the Java Tutorial. Which includes a lesson All About Sockets. This example is from that tutorial.

```
try (
    ServerSocket serverSocket = new ServerSocket(portNumber);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
        new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
) {
    ...
    out.println(outputLine);
    while ((inputLine = in.readLine()) != null) {
        outputLine = processInput(inputLine);
        out.println(outputLine);
        ...
    }
```

Guessing Game Server

Requirements

Non-Requirements

Your Code

- Java
- **Python**

Python

An excellent starting place for **Python** is the Python Documentation. The Socket Programming HOWTO is very good and walks through an object-oriented approach to socket programming in Python. This is a much simpler example.

```python
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('localhost', 2222))
s.listen(10)
conn, addr = s.accept()
conn.send('hello'.encode('utf-8'))
while True:
    data = conn.recv(1024)
    line = data.decode('utf-8')
    response = process_data(line)
    conn.send(response.encode('utf-8'))

s.close()
```

Guessing Game Server

Requirements

Non-Requirements

Your Code

- Java
- Python
- **Other**

Other Languages

Most other modern and general purpose programming languages provide for **socket programming** in some way. If you use a different language, you will have to find your own introduction materials.

You should also be aware that, **I need to be able to compile and run your code**.

You should choose a language that is available on the CS Lab UNIX/Linux systems. That is where I will test your code.

Known working languages: `Python`, `Java`, `Rust`, `C/C++`. Talk to me if you are thinking of using something other than these.