CS2044 Homework 3

Due: Sunday March 20th 2011 at 11:59 PM on `http://cms.csuglab.cornell.edu`.

General Note: This assignment (and future ones) require you to have access to a Unix-like (Linux, Mac OS X, etc) machine. If you do not have such an operating system installed on your local machine, make sure to get a CSUG Lab account. Different systems have slightly different configurations. The main environment in this class will be GNU/Linux.

Assignment Notes:

- You must complete the assignment using GNU/Linux tools that were discussed in class.
- Complete the assignment by submitting your script on CMS.
  - plus a simple feedback on the assignment

This assignment may seem hard, but it is not.

In this assignment I want you to gain experience writing scripts that interact with others on the network. We'll do this by implementing a networked game. However, I am not interested in the game's implementation. In fact, I will point you to some source code that does that. However, I do want you to learn to write scripts that send and receive messages on the network.

What is the assignment?

Your objective in this assignment is to write a python script, `./moo.py`, that can play the game of *Bulls and Cows* with an opponent on the network. This is a two player game, where each player tries to guess a secret 4-digit number that the opponent has. A player's secret number is composed of four unique digits. So for example, 1234 is a valid secret number, but 1123 is not.

Gameplay

The game proceeds in rounds, in each round a player makes a guess for the opponent's secret, the opponent responds back with how close the guess was. More specifically, when a player

makes a guess, the opponent will tell the player how many digits of the guessed number matched the secret and were in the correct position (called *bulls*), and how many digits matched the secret by were in an incorrect position (called *cows*). Examples:

- For secret 1234, guess 1873 has: one bull (the digit 1) and one cow (the digit 3)
- For secret 1234, guess 1384 has: two bulls (the digits 1 & 4) and one cow (the digit 3)
- For secret 1234, guess 6789 has: zero bulls and zero cows

Each player tries to assemble the bulls/cows data in order to figure out the other player's secret number. The first player to guess the secret number of the other wins.

You can read more about bulls and cows here:
`http://en.wikipedia.org/wiki/Bulls_and_cows`

Network interaction

Now that we talked about how the game works on pen and paper, we need to figure out how to make two programs play it on the network.

In order for two programs to understand one another, they have to agree on the format of messages they're going to exchange, where are they going to listen for messages, and how will the messages be sent to them. This information is described in what is referred to as a *communication protocol*.

First, where to look for messages? Each program on the Internet is identified by an *IP-address* and a *port number*. An IP-address is much like a building address in that it is shared by all the people who reside in that building. A port number is like the individual mailboxes where different people in the same building will differentiate their mail by their mailbox numbers. So, for an opponent to be able to send you messages, it has to know your machine's IP address and the port number on which your program is listening.

Second, how to actually send messages? The physical transport of bits from one location to another (i.e. from your program to another, or from your web browser to the server responsible for `example.com`) is handled by a *transport protocol*. There are two main transport protocols, and they are the ones used on the Internet: TCP and UDP. In this assignment we'll use UDP.

Third, the format of the messages. Here we get define our own standard format that both players know and agree on. We'll define the format of the message next, and the game's protocol next.

For this assignment, we will follow a very simple protocol:

- Your program will send and receive messages via the UDP transport protocol (a simple protocol to exchange discrete messages on the Internet)

- When your program starts up it will be given the port number on which to listen for incoming messages, and the port number on which to send messages to the opponent. We will assume that the opponent is also on the same machine.
- Each message is in one of the following formats:
  - `GUESS:`$wxyz$
    This message contains a player's guess.
    Examples: `GUESS:1234` , `GUESS:4567` , `GUESS:3845`
  - x`B`y`C`
    This message means that the previous guess had x bulls and y cows.
    Examples: `1B1C` , `2B1C` , `0B3C` ..etc
  - `WIN`
    This message means that you guessed the secret number correctly

Your script

- You're script will require 3 arguments when it starts: the secret, it's port number, and the opponent's port number. For example, to start with secret 1234, and local port 4000 and opponent port 4001:

```
./moo.py 1234 4000 4001
```

- When the script starts, it will start making guesses for the opponent's secret. It will print out each guess to the screen followed by what the opponent responded with.
- Your program will terminate when either it won or the opponent won.
- You can test your script by launching two instances of it and having them play against one another:

```
./moo.py 1234 4000 4001
./moo.py 9876 4001 4000
```

How to implement this

There are three parts to this assignment:

- **Checking a guess against the secret number and calculating the proper response**. This is not the focus of this assignment, and you can indeed find an implementation for this here: `http://rosettacode.org/wiki/Bulls_and_Cows#Python`

- **Sending and receiving messages on the network**. This is really two parts, the *server* which handles incoming guesses from the opponent, and the *client* which issues guesses to the opponent.

  The following page from the python documentation has a good example how to write UDP client/server loops in Python:
  `http://docs.python.org/library/socketserver.html#socketserver-udpserver-example`

- **Making good guesses**. Your program can blindly make guesses until it stumbles upon the correct secret number, or it can try to learn something from what the opponent responds with. This part is completely optional, and I'll leave it up to you. It would be cool to implement some simple calculations to reduce the number of guesses.

Notes:

- Test your program by running two instances of it against each other on your machine (the same machine) as I showed earlier.
- If you get stuck, there is a wealth of information on how to write network programs in Python on the web.

Remember your two best friends are your favorite search engine, and the Python documentation: `http://www.python.org/doc/`

Good Luck!