


Intro to Nets 2023/4 Alternative Evaluation For Reservists - Trivia King

Version 1.0

Introduction

Your objective is to write a client-server application which will implement a trivia contest. The players in this game get a random fact, which is either true or false, and are supposed to answer correctly as fast as possible.

Each team will write both a client application and a server application. Everybody's clients and servers are expected to work together with full compatibility.

 If you are in a team of 3, you will also be expected to write a “bot” application which behaves like the client.

Example Run

1. Team Mystic starts their server. The server prints out “Server started, listening on IP address 172.1.0.4” and starts automatically sending out “offer” announcements via UDP broadcast once every second.
2. Players Alice, Bob and Charlie start their clients. The clients all print out “Client started, listening for offer requests...”. All of the clients get the offer announcement, extract the server name, and print out “Received offer from server “Mystic” at address 172.1.0.4, attempting to connect...”
3. Each client connects to the Team Mystic server over TCP using the port specified in the ann. After the connection succeeds, each client sends the player name over the TCP connection, followed by a line break ('\n')
4. After 10 seconds pass during which no additional player joins, the game begins - the server sends a welcome message to all of the clients with the names of the teams, followed by a statement which is either true or false. The answer to the question should be either Y,T,1 for true statements, or N,F,0 for false statements:

```
Welcome to the Mystic server, where we are answering trivia questions about Aston Villa FC.
```

```
Player 1: Alice
```

```
Player 2: Bob
```

Player 3: Charlie

==

True or false: Aston Villa's current manager is Pep Guardiola

5. Every time the server sends the client a message over TCP, the client immediately prints it to the screen. The server also prints out this message to its own display. Every time the player types in a key, the client sends it to the server over TCP
6. Each time the server receives a key over TCP, it checks the correctness of the answer:
 - If the answer is correct, the player wins immediately
 - If the answer is incorrect, the player is disqualified
 - If nobody answers after 10 seconds, or if all players answered incorrectly and were disqualified, the server chooses another random trivia question.

If you are in a team of 2, the game finishes after 1 round, and the user who answers the correct answer first wins.

👉 If you are in a team of 3, the game continues for multiple rounds which are played between all users who answered correctly within 10 seconds, until only 1 player is left standing, and this player wins the game.

Example output for team of 2:

Alice is correct! Alice wins!

👉 Example output for team of 3:

Alice is correct!

Bob is incorrect!

Charlie is correct!

Round 2, played by Alice and Charlie:

True or false: Aston Villa's mascot is a lion named Hercules

Alice is incorrect!

Charlie is correct! Charlie Wins!

7. After the game is decided, the server sends a summary message to all players, for example:

Game over!

Congratulations to the winner: Charlie


8. The server closes the TCP connection, prints on its own screen "Game over, sending out offer requests..." and goes back to sending offer messages once a second
9. The clients print "Server disconnected, listening for offer requests..." and go back to waiting for offer messages

Suggested Architecture

The client is a single-threaded app, which has three states:

- Looking for a server. You leave this state when you get an offer message.
- Connecting to a server. You leave this state when you successfully connect using TCP
- Game mode - collect characters from the keyboard and send them over TCP. collect data from the network and print it on screen.

Note that in game mode the client responds to two events - both keyboard presses and data coming in over TCP. Think about how you can do this.

 For teams of 3, you should also implement a “bot” client. The bot behaves the same way as the client, but it randomly chooses answers instead of listening to the keyboard. It should be possible to run an unlimited number of bots in parallel. Try to engineer your code so that there is code reuse between the bot and the standard client.

The server is multi-threaded since it has to manage multiple clients. It has two states:

- Waiting for clients - sending out offer messages and responding to request messages and new TCP connections. You leave this state after 10 seconds pass from the last time a user joined the game.
- Game mode - collect characters from the network and decide the winner. You leave this state after somebody wins the game.

Packet Formats

- Servers broadcast their announcements with destination port 13117 using UDP. There is one packet format used for all UDP communications:
 - Magic cookie (4 bytes): 0xabcdcdca. The message is rejected if it doesn't start with this cookie
 - Message type (1 byte): 0x2 for offer. No other message types are supported.
 - Server name (32 character string, Unicode encoded): Think of a creative name. Make sure this field is always 32 characters long even if your server name is shorter.
 - Server port (2 bytes): The port on the server that the client is supposed to connect to over TCP (the IP address of the server is the same for the UDP and TCP connections, so it doesn't need to be sent).
- The data over TCP has no packet format. After connecting, the client sends the predefined team name to the server, followed by a line break ('\n'). After that, the client simply prints anything it gets from the server onscreen, and sends anything it gets from the keyboard to the server.

Tips and Guidelines

- Please pick a creative name for your players. The player name should be hard-coded into the program so it will start up quickly. 🗨️ For teams of 3, the bot's team name should be randomly generated whenever the bot starts, but it should start with "BOT:"
- ChatGPT and Github Copilot are both allowed.
 - Please create a dedicated "chat" in the ChatGPT interface for this assignment, and submit a transcript of this chat together with your assignment for grading.
 - It's very easy to use ChatGPT to generate trivia questions. They don't have to be about football, you can choose any topic you like! Here is the query I used:
"Please create a Python list of 20 random trivia questions about Aston Villa FC, both true and false, together with an "is_true" field for each question"
- Both server and client applications are supposed to run forever, until you quit them manually. You will not get full points if your program quits or crashes, even if it's because of some network-related problems.
- The server does not have to listen on any particular port over TCP, since this is part of the offer message. Think of how your code should respond if the first port you try to listen on is already used by somebody else
- **Do not use busy-waiting (e.g. while-continue loops).** As a general guideline, if your program consumes more than 1% CPU, you're probably doing something wrong.
- Think about what you should do if things fail - messages are corrupted, the server does not respond, the clients hang up, etc. Your error handling code will be tested.
- If you try to run two clients on the same computer, they won't be able to listen for broadcasts on the same UDP port unless you set the SO_REUSEPORT option when you open the socket, like this:

```
>>> s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
>>> s.bind(("",13117))
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

OSError: [Errno 98] Address already in use

```
>>> s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
```

```
>>> s.bind(("",13117))
```

- The assignment will be written in Python 3.x. You can use standard sockets or the [scapy package](#) for network programming. The [struct.pack](#) functions can help you encode and decode data from the UDP message.
- The University Wi-Fi network blocks broadcast messages, so you won't be able to use it for development. The best option is to use a phone hot-spot.
- Please set up a git repository for your own code and make sure to commit regularly. Visual Studio Code has git support so it's quite easy.

To Get Full Points on Your Assignment

- Work with any client and any server
- Write high-quality code (see below)

- Have proper error handling for invalid inputs from the user and from the network, including timeouts
- Bonus: use [ANSI color](#) to make your output fun to read
- Bonus: collect and print interesting statistics when the game finishes (best team ever to play on this server, most commonly typed character, anything else you can think of...)

Code Quality Expectations

How to Submit

Please submit to the Moodle a link to your github repository (make sure it's not private). You can commit as much as you want, but only the last commit before the deadline will be considered.

Static Quality

- Code has proper layout, and meaningful function and variable names
- Code has comments and documentation for functions and major code blocks
- No hard-coded constants inside code, especially IP addresses or ports

Dynamic Quality

- Return values of functions are checked
- Exceptions are handled properly
- No busy-waiting!
- Network code is isolated to a single network

Source control

- Code is hosted in a github repository
- Commits were made by all members of the team
- Proper use of commit messages and branches

Good luck! 🤞