Assignment 1: Protocols and Sockets (10%)

Fall 2023 – CPSC 441

Due at 23:59, Sep. 28 on D2L

Assignment policy:

- This is an **individual** assignment, so the work you hand in must be your own. Any external sources used must be properly cited (see below).
- Extensions will not be granted to individual students. Requests on behalf of the entire class will only be considered if made more than 24h before the original deadline.
- Some tips to avoid plagiarism in your programming assignments:

  1. Cite all sources of code that you hand in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

     ```
     # the following code is from https://www.quackit.com/hello_world.cfm.
     ```

     Use the complete URL so that the marker can check the source.
  2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
  3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
  4. Collaborative coding is strictly prohibited. Your assignment submission must be entirely your code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. You cannot use another student's code, even with citation.
  5. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task.
  6. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

Background:

Fred is dead. Everyone is dead. During one of your scavenging missions as the sole survivor of the nuclear apocalypse, you've found a worn-out server with some unknown program on it. The barely legible label reads "Tic Tac Toe" - a meaningless phrase in a strange language. After some preliminary testing you've figured out this server is running a game of sorts, but it needs a client to function. Using your expert coding skills, you set out to make a client, allowing yourself to reminisce about an age long past.

The goal of this assignment is to familiarize yourself with application-layer protocols and socket progamming. To complete the assignment, you will be building a client that can interact with a server according to the protocol specified below. You will need to create a game loop that can send and receive data across a socket connection in order to play a game of Tic Tac Toe against the sever.

Note on languages:

You may choose to complete this assignment in either C/C++ or Python. The assignment description has been written with Python in mind, and learning to use sockets in a language you already know (like Python) helps avoid the extra barrier of coding in a new language. I find it is often helpful to see these things in more than one language, plus having some experience with Python sockets will likely prepare you better to use networking in your personal projects. That said, you are welcome to use C or C++ if you are comfortable. The practice with C/C++ will definitely be useful with preparing for your future assignments in this course.

Instructions:

The server for this assignment can be found on the university cpsc server `csx.cpsc.ucalgary.ca` (IP `136.159.5.25`) on port `6969`. If this server goes down for any reason, you can also run the server locally. See the tips for more details.

The server handles keeping track of the game state and making moves for the AI opponent. Your client needs to handle its own main game loop and board rendering, requesting data from the server where appropriate.

There is a lot of freedom in how your client functions, but at minimum it requires the following features:

- The client must be contained in a single file called `tictacClient`. Your code should take two command line arguments, where the first argument specifies the IP address of the server and the second argument specifies its port number.
- When the client is started, it should display a welcome banner and offer the following menu options:

  - New game
  - Load saved game
  - Show score
  - Exit

- Each turn, your client should render the current game board and allow the human player to choose a move. You will need to communicate with the server to obtain its move in response, and then update the display accordingly.

- Your client needs to implement functionality for loading and saving games. When the user selects either the load or save option, the user should be prompted for a filename. Your save game files should be stored in a hard-coded subdirectory of the location your client script is run from. Make sure you include code to create this directory if it does not already exist – remember that the TAs will be downloading and running your client on the university computers.

- You need to have proper error handling for any networking errors that may occur (e.g. the client cannot connect to the server, or the socket connection is terminated)

To submit your assignment, make sure to upload both your source code (a single file named `tictacClient.py`, `tictacClient.c`, or `tictacClient.cpp`) and your README to the assignment dropbox on D2L.

Network protocol:

To interact with the server, the client sends messages of the form `HEAD:BODY`, where `HEAD` is the four-character header code and `BODY` is the body of the message. Note that some messages do not have a body, in which case the colon can be left off.

Below is a list of all messages used in the protocol:

- `NEWG`: This message has no body. It causes the server to initialize a new game board and randomly determine if the CPU or player moves first (remember, X always goes first). The reply is a `BORD` message.

- `ENDG`: This message has no body. It causes the server to end the current game and reset the board. The reply is an OVER message with the body as "no winner" and the current state of the board.

- `CLOS`: This message has no body. It causes the server to close the connection to the client. The reply is another `CLOS` message.

- `MOVE`: This message is sent to the server during an active game to signify the player has moved. The body is the coordinates of the move in the form of [row],[column]. For example, `MOVE:1,1` places the player's move in the middle of the 3x3 grid. The server handles error checking and will return an `EROR` message if there is an error. If

there is no error, the server will return either an `OVER` message if the game is over or a `BORD` message if the game is still ongoing.

- `BORD`: This message is sent to the client when the board has been updated (i.e. after a move or a load). The body consists of nine comma-separated integers indicating the board state. In this list, 0 represents X, 1 represents O, and 2 means the space is empty. For example, `BORD:2,2,2,2,0,2,2,2,2` means that X is in the middle space and the rest of the board is empty.
- `LOAD`: This message is sent to the server to load a saved game. Recall that the user should provide a filename with the saved game when selecting this menu option. The body of this message is the character of the human player (X or O), followed by a comma and then nine comma-separated integers representing the board state as described above. For example, `LOAD:X,0,1,0,2,1,2,1,2,0` loads a game board where the client is X and each player has made three moves. The reply is a `BORD` message with the loaded game state.
- `EROR`: This message is sent by the server to indicate something went wrong. The body is the error descriptor, which has three possible values:
  - `UNKNOWN CMD` is sent when the message header sent by the client is unknown
  - `BAD MOVE` is sent when the `MOVE` message contains an invalid move
  - `NO GAME` is sent when the client is trying to `MOVE` before a game has started.
- `OVER`: This message is sent by the server to indicate the game is over. The body contains the winner (if there is any) and the current state of the board. C means the client won the game, S means the server did and N means no one won. For example, `OVER:C,0,0,0,1,1,2,2,2,2` is sent when the client won playing as X.

Rubric (40 points total):

- **20 marks** for having a functioning main menu and game loop, allowing a user to play a game with the server. Your code should communicate with the server as outlined in the assignment description.
- **10 marks** for correctly implementing the save and load functionality.
- **5 marks** for correctly handling potential errors, including bad user inputs.
- **5 marks** for having clean and well-documented code, including a README file with a description of how to run your code and listing any known bugs.

Note that marks will be deducted if the implemented functionality deviates from the assignment specifications.

Tips:

- If the hosted server seems to be down, or you want to test things locally, you can run the server locally by downloading the binary from D2L. The server was compiled for Linux x86-64, so make sure you run it on a machine (or VM) with this OS and architecture. You can run it with the following command:

  ```
  ./tictactoeServer [PORT]
  ```

  Here, `PORT` is an optional argument specifying which port the server should bind to. If none is provided, it will default to `6969`. After being invoked, the server will wait and listen for incoming connections.

- Try simply printing out the raw packets you receive, as it will help you get comfortable with how the server behaves.

- You can use utilities like telnet to send custom packets directly to the server without building a Python client. Make sure you are comfortable with how it works, and use it prolifically for testing and debugging. Don't be afraid to send arbitrary packets to see how the server responds. Remember, this server is your only friend after the apocalypse, so you might as well get familiar with each other.

- I've hidden a secret message that will only be sent by the server if you manage to win against its AI. There will be a reward for the first person that sends me a screenshot proving that you received it.

---

author: jcleahy@ucalgary.ca