MPCS 52040 Distributed Systems
Homework 1: Tic-Tac-Toe
Due: 1/17 @ 10:00am

Client-server Tic-Tac-Toe

In this homework assignment you will create a client/server Tic-Tac-Toe game. The game will allow clients (invoked individually) to place markers in a game board maintained by the server. The server will ensure moves are permitted and determine a winner of the game.

The project should be implemented in Python 3 and use the standard Python socket and pickle libraries.

You should implement the client and server command line interfaces exactly as described below to enable straightforward grading.

Description

The server must store the state of the Tic-Tac-Toe game (i.e., a 3x3 matrix). The server should validate moves (e.g., that the position is open, that it is the correct turn) and declare a winner (or draw) at the conclusion of the game. After every move the server should evaluate the board to determine if there is a winner.

The server should be implemented using sockets. It should first create/bind to a socket and then listen for messages in a loop. It should be started with the following command:

```
Python server.py <port>
```

The client should be implemented as a Python program that connects to the socket and sends messages (moves) and receives responses (game board, errors, winner status). The responses should be sent as **serialized messages** rather than strings (see the section below).

The client should be invoked as follows by including the player and the position of the move.

```
Python client.py <port> <player> <coordinate>
```

Represent a player as "O" or "X" and a move as a coordinate pair (row, column). Where row and column should be zero relative (i.e., 0–2). The game may start with either player going first. After each move the server should respond with the updated game board, an error message if the move is invalid, or declaration of the result (winner/draw).

For example, if the Player "O" wants to place their marker in the top right corner they would invoke the client as follows.

```
Python client.py <port> O  "(0,2)"
```

The server should process the move and respond with the gameboard. The client should then print the state of the game, for example:

```
O | X | O
X |   | X
```

```
O |   |
```

If the client places their marker in an invalid position the server should reply with an error, for example:

```
Move (0,2) is invalid because there is already a marker in that position.
```

Consider other types of errors that could occur and ensure your server can handle those errors.

In the final turn of the game (e.g., for draw or win), the server should reply (using your message format) with the result and include the gameboard. The client should accept that object and print the result and the gameboard (i.e., the object must be passed between client/server), for example:

```
Player O has won the game.

  O | X | O
  X | O | X
  O |   |
```

Note: the server must maintain the state of the game until it is restarted.

You should add a method to restart the game as follows.

```
Python client.py <port> restart
```

When the restart command is received, the server should remove it's current state.

Serialization and exchange protocol

You should develop a protocol to communicate the players' moves, the gameboard, restarts, and any errors. You should create objects to represent the messages and use Python pickle to serialize these objects to be passed over the socket.

You will need to create custom objects to represent requests (i.e., moves) and responses (i.e., the gameboard, errors, and the winner). The objects can be simple (e.g., an error could be represented as an error class with a message) but must not be a standard Python data structure (i.e., you should define your own classes). The objects should then be serialized using Pickle and sent over the socket. The receiving side should deserialize the pickled message and perform the action or display it to the user.

Submission

GitHub classroom link: `https://classroom.github.com/a/l7oCaqNe`

When you first login you will need to "Join the classroom" and select your UChicago email address so that we can map your GitHub account to your CNet ID. You should see a message like this:

"To join the GitHub Classroom for this course, please select yourself from the list below to associate your GitHub account with your school's identifier (i.e., your name, ID, or email)."

Select your email address from the list and continue.

You will then be asked to accept access to the homework 1 repository

You should now be given a private GitHub repository in the classroom. You can commit changes to your repository. We will take the last commit to the main branch before the deadline as the final version of the homework.