CS 4500 Project

Phase 6: Client/Server

Due: Monday, November 25, 11:59pm

Submission:

For the *Design Task*: place `protocol.md` in the `Planning/` directory.

For the *Programming Task*:

- Place your new strategy implementation in `Player/`
- Place your server and client files in a new directory called `Remote/`

For the *Testing Task*, place the following in a new directory `6/` within `Tsuro`:

- `xserver`
- `xclient`
- `xrun`
- `README-tests.md`, describing how to run the test harnesses.

Design Task

Specify a TCP based protocol for the game, including the connection of individual players, the start of the game, the referee-players interaction during the game, and the end. Use the referee-player interaction diagram from Phase 4 as a starting point and specify the **exact syntax of messages** and their ordering, for each phase of the game.

Programming Task

Client/Server Implementation

At this point you should have a local implementation of a Tsuro game as an interaction between components representing the game system and a player component. The purpose of this task is to take these components and implement a client-server architecture for the game, with the Tsuro game server on one end and 3-5 automated player clients on the other. Use the protocol you specified in the Design Task as the communication protocol between the server and the players.

Since you already have an implementation of the game components, as well as the player, and you tested an integration of these, we recommend using the *remote proxy* pattern. Following this pattern, implement a player proxy module on the server side. The proxy implements the player interface, but uses a TCP socket connection to pass the requests to the remote player implementation. Responses from the player are processed and returned as a result of the respective method call. The referee does not (should not) need to know that the players are just proxies. On the player side, implement a layer, which, after connecting to a game server, will process requests from the server and call the corresponding methods of the *actual* player implementation. The results of the calls are passed back to the server.

The server should take two optional parameters: an IP address (or hostname) and a port. The defaults for these parameters are `127.0.0.1` (`localhost`) and `8000`. If only one parameter is given, it is the port. The server should listen on the specified hostname/port for player client connections. The minimum number of players is 3. After the first three players connect, the server should wait for another 30 seconds for two more players. Once 3 to 5 players connect, the server should refuse any additional connection and should start-up a Tsuro game between a referee and the connected players.

The client should take four mandatory parameters: an IP address (or hostname), a port, a player name, and a strategy. It should connect to the game server at the given address and port, and wait for requests (messages) from the server.

Note: If you search online for suggestions on how to handle multiple client connections, most tutorials seem to suggest a multi-threaded solution, where each connection is handled by a separate thread. For this task, using multiple threads shouldn't be necessary. If you have a hard time making it work, we have some example code in Python that might be helpful. However, we'd like you to try figure it out yourself first.

Optional: you can also turn the observer into a remote client. If you do, account for this in your protocol specification in `Planning/protocol.md`.

Strategy

To make players more interesting, implement at least one more player strategy. You can implement your own strategy, or you can implement the one described below (named `Second/second`). Or you can do both.

1. When the player is asked to place an initial tile, it searches for the first legal spot available in counter-clockwise direction starting from (0,0) [exclusive]. To place the avatar, the player searches for the first legal port in counter-clockwise fashion that faces an empty square. The player uses the third given tile, without rotating it.
2. When the player is asked to take its turn, it starts the search for a legal option with the second tile type, trying all possible rotations starting from 0 degrees. If none of these possibilities work out, it goes back to the first one and repeats the process. If no possible action is legal, it chooses the second tile at 0 degrees.

If you implement your own strategy, state its name and describe how it works in a document called `6/strategy.md`.

Testing Task

Provide the server and player executables, `xserver` and `xclient` in `6/`. These executables should take the arguments as specified above. The server executable should log all communication between the server and the players to a file called `xserver.log`. Each message should be prepended with the player's color and whether it was sent or received (some kind of arrows, like >> and <<, suffice for expressing the direction of communication).

Additionally, write a program or script, called `xrun`, which reads a JSON array of player specifications from standard input. The array has the following format:

```
[ { "name" : String, "strategy" : String}, ... ]
```

where the `"name"` field is the player's name, and the **case insensitive** string in the `"strategy"` field is the selected strategy for that player. At least `"dumb"` needs to be supported). The array will contain 3-5 such player specifications. Inputs with less than 3 or more than 5 elements should cause the program/script return an error.

Once input is processed, the program should start up the server and the corresponding number of clients, locally, instructing the clients to connect to the server's address and port.