NSL
Networked Systems Lab

CAU
CHUNG-ANG UNIVERSITY

Computer Network

Applications & Design

Homework Assignment #5

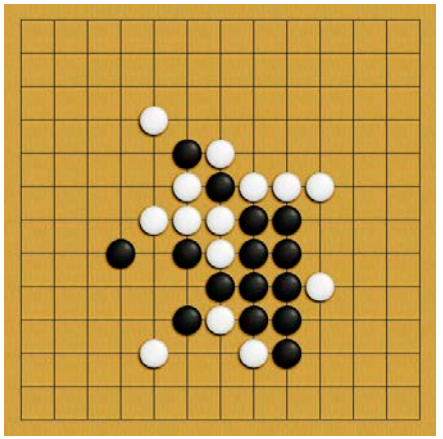
P2P Omok (5+2 points)

Due date: **June 5th (Sun) 2022, 11:59PM**.
- submit softcopy on class server

Overview

- **In this assignment, you will design and implement peer-to-peer (P2P) omok game application.**
  - Although it's P2P, there is a 'game rendezvous server' which arranges two client peers to meet and play with each other.
  - Once two peers get to know of each other, they play omok game *'directly'* with each other peer-to-peer without involving the server. They can also chat with each other.
  - You will use both TCP and UDP, and **Go** language for this homework assignment.

- **'Omok'** is a simple board game
  - Two players (black and white) take turns, and place a stone on a grid at each turn.
  - You win if you can get five of your stones in a row (straight line).



✓ All Korean students know what 'omok' game is. However, there should be no disadvantage to non-Korean students. Thus, to make things fair, I'll provide you a Python source code for stand-alone non-networked version of the omok game ('omok.py'). This will give you a hint on how to implement the basic 'board print' function and 'win/lose detection' function.

✓ No fancy GUI is required; text output is good enough.

Example Screen Output

```
$ go run P2POmokServer.go

ironman joined from 165.194.35.202:34567. UDP port 23456.
1 user connected, waiting for another

superman joined from 165.194.35.202:24680. UDP port 78901.
2 users connected, notifying ironman and superman.
ironman and superman disconnected.
```

```
$ go run P2POmokClient.go ironman

welcome ironman to p2p-omok server at 165.194.35.202:59999.
waiting for an opponent.



superman joined (165.194.35.202:78901). you play first.

  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + + + + + + + |
4 | + + + + + + + + + + |
5 | + + + + + + + + + + |
6 | + + + + + + + + + + |
7 | + + + + + + + + + + |
8 | + + + + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------

hi what's up?
superman> don't you think prof. Paek is handsome?
couldn't agree more.
anyways, let's play omok
supernam> okay, it's your turn

\\ 3 4

… ~~~ next page… ~~~ …
```

```
$ go run P2POmokClient.go superman

welcome ironman to p2p-omok server at 165.194.35.202:59999.
ironman is waiting for you (165.194.35.202:23456).
ironman plays first.

  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + + + + + + + |
4 | + + + + + + + + + + |
5 | + + + + + + + + + + |
6 | + + + + + + + + + + |
7 | + + + + + + + + + + |
8 | + + + + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------

ironman> hi what's up?
don't you think prof. Paek is handsome?
ironman> couldn't agree more.
ironman> anyways, let's play omok
okay, it's your turn

… ~~~ next page… ~~~ …
```

Example Screen Output – Client/Peer

```
  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + O + + + + + |
4 | + + + + + + + + + + |
5 | + + + + + + + + + + |
6 | + + + + + + + + + + |
7 | + + + + + + + + + + |
8 | + + + + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------
```

… ~~~ skipping… ~~~ …

\\ 5 6

```
  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + O @ + O + + |
4 | + + + + + O @ O + + |
5 | + + + + O @ O + + + |
6 | + + + + @ + + + + + |
7 | + + + @ + + + + + + |
8 | + + + + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------
```

superman> haha! I’m gonna win!
oh, shit

```
  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + O @ + O + + |
4 | + + + + + O @ O + + |
5 | + + + + O @ O + + + |
6 | + + + + @ + + + + + |
7 | + + + @ + + + + + + |
8 | + + + + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------
```

… ~~~ skipping… ~~~ …

```
  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + O @ + O + + |
4 | + + + + + O @ O + + |
5 | + + + + O @ O + + + |
6 | + + + + @ + + + + + |
7 | + + + @ + + + + + + |
8 | + + + + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------
```

haha! I’m gonna win!
ironman> oh, shit
\\ 8 2

Example Screen Output – Client/Peer

```
  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + O @ + O + + |
4 | + + + + + O @ O + + |
5 | + + + + O @ O + + + |
6 | + + + + @ + + + + + |
7 | + + + @ + + + + + + |
8 | + + @ + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------
you lose

gg man!
superman> good game.
superman> bye, and see you next time!

\exit
Bye~
```

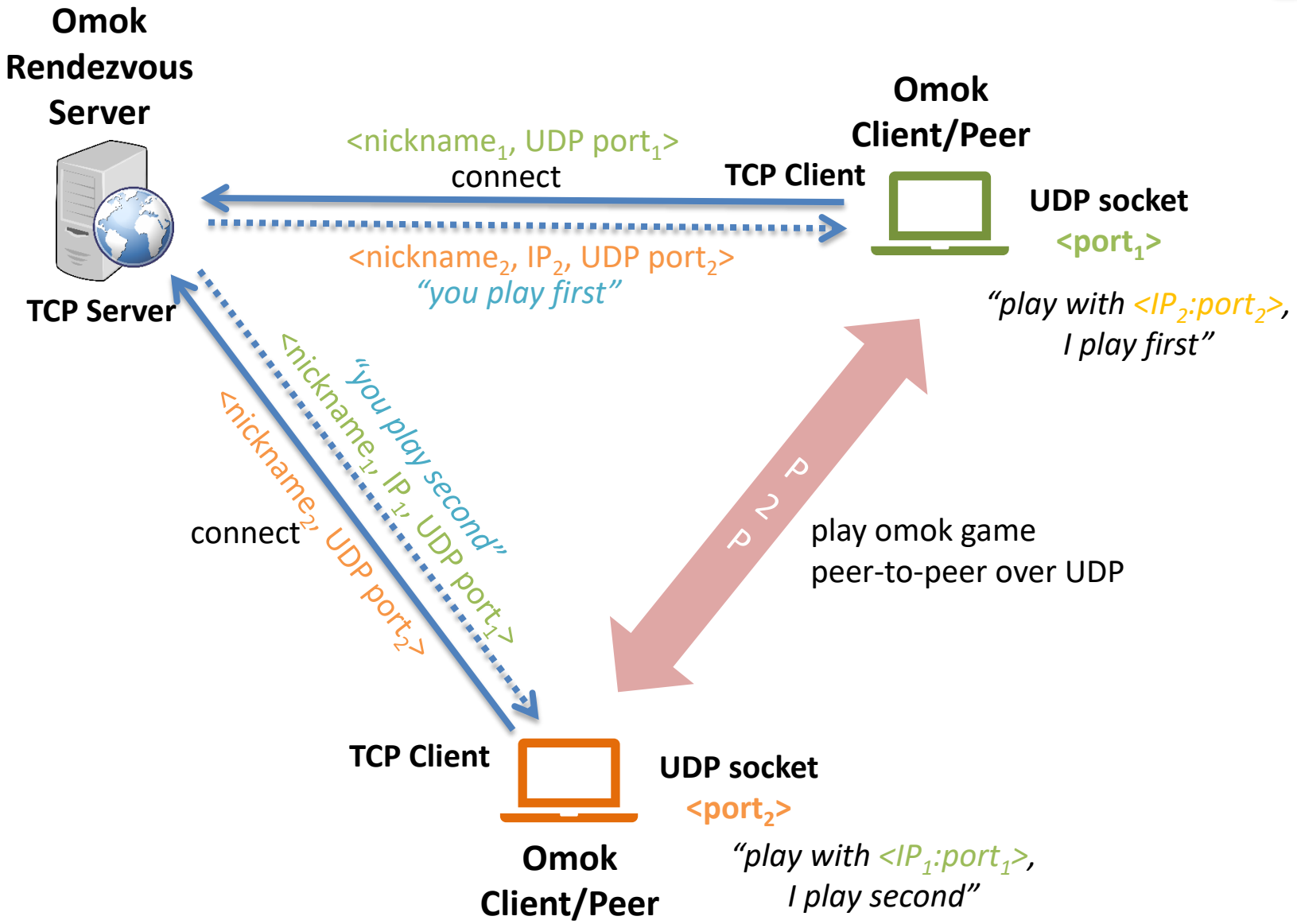
```
  y 0 1 2 3 4 5 6 7 8 9
x ----------------------
0 | + + + + + + + + + + |
1 | + + + + + + + + + + |
2 | + + + + + + + + + + |
3 | + + + + O @ + O + + |
4 | + + + + + O @ O + + |
5 | + + + + O @ O + + + |
6 | + + + + @ + + + + + |
7 | + + + @ + + + + + + |
8 | + + @ + + + + + + + |
9 | + + + + + + + + + + |
  ----------------------
you win

ironman> gg man!
good game.
bye, and see you next time!

\exit
Bye~
```

Components

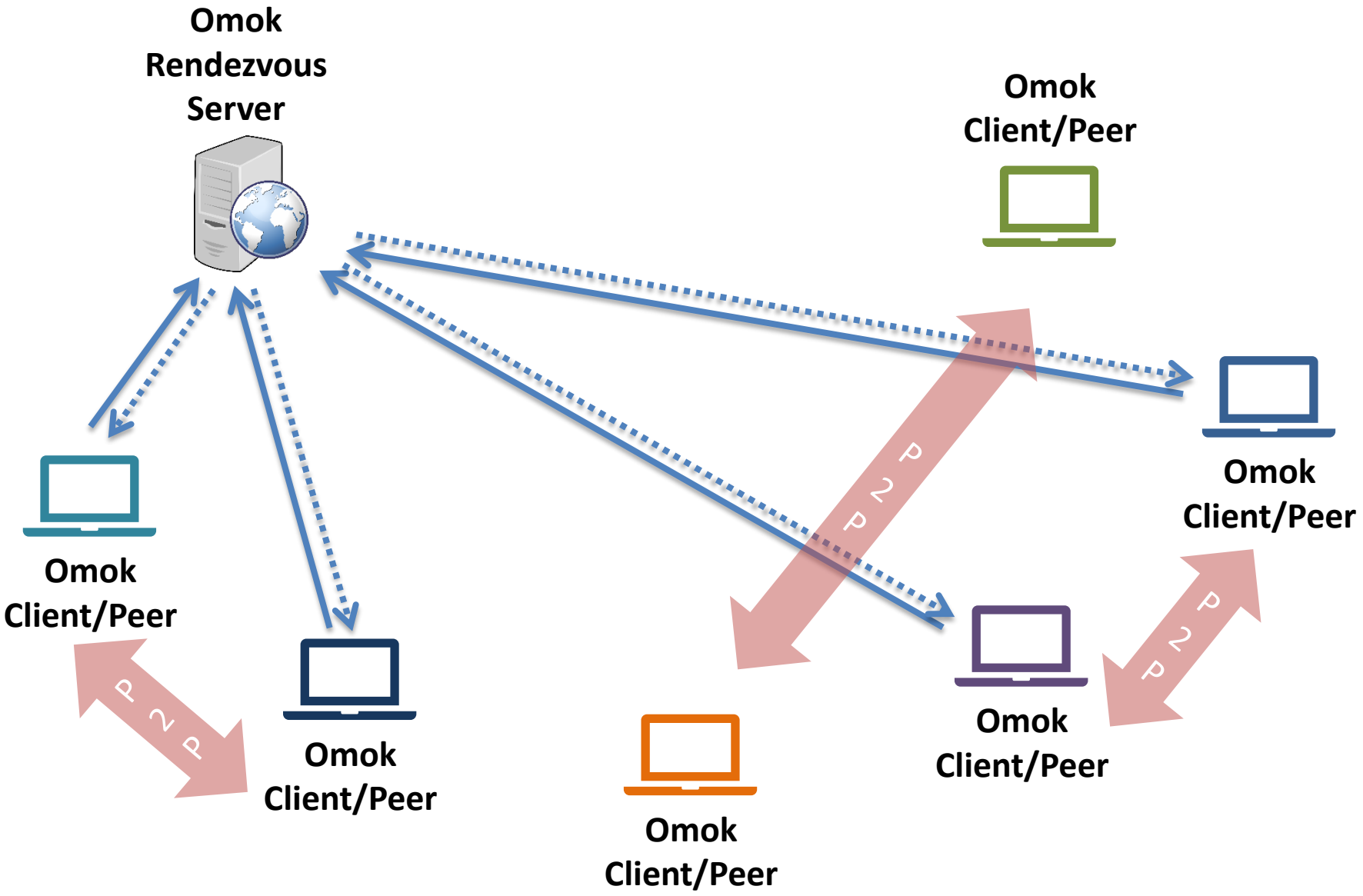
- **Rendezvous Server**
  - TCP server, accepts connection from clients that wish to play.
    - When there is 0 or 1 client connected, it waits.
  - As soon as there are 2 clients (immediately when 2nd client connects),
    - It notifies both clients about the information of each other,
      - ✓e.g. IP, UDP port number, nickname, who plays first
    - triggers them to start the game with each other, and
    - spin-off the two clients (i.e. disconnects them).

- **Client/Peer**
  - UDP peer for playing omok game.
    - Use UDP socket, and communicate directly to the peer for playing the game.
  - TCP client for finding a game opponent on the server.
    - At boot up, connect to rendezvous server and notify it's IP, UDP port number (and nickname).
    - When notified by the server regarding its game opponent,
      - ✓e.g. IP, UDP port number, nickname, who plays first
    - disconnect TCP socket and start playing on UDP socket.

Basic Architecture



When there are more users



Implementation

- You must implement two programs,
  - **"P2POmokServer.go"** for the rendezvous server, and
    - TCP server port number should be 5XXXX (one of your personal designated port number)
  - **"P2POmokClient.go"** for the client/peer.
    - Client should take one command line argument, <nickname>.
    - Rendezvous server address <IP:TCPport> should be hardcoded in client code.
    - For TCP client, do not specify port number (use random number assigned by OS)
    - For UDP socket, do not specify port number (use random number assigned by OS)

- You will use…
  - **TCP** socket
    - for communicating with the server to find a game opponent; disconnect when done.
  - **UDP** socket
    - for playing the game with the opponent in a peer-to-peer manner.
    - Program terminates when the game is over (or Ctrl-C is pressed).
    - Connection management is handled at the application layer.

Chat Message and Commands

- Client can set its nickname using a command line argument.
  - For example, when I run my client program, I can set my nickname as "*ironman*" and you can set yours as "*superwoman*." Then these will be shown for all chatting.
  - You may assume max. length of ≤ 64 bytes ASCII for a nickname, English nickname, and no space or special character (e.g. '\') in the nickname.

- While chatting, user can type text message and press `Enter` to send that message.
  - that message should be delivered to your opponent peer.

- Chat room commands:
  - **\\ <x> <y>** // places a stone on the <x> <y> coordinate of the game board.
  - **\gg** // gives up the game
  - **\exit** // you notify the peer that I'm exitting, and quit
  - else, *"invalid command"*.

Command details

- \\ <x> <y>
  - Places a stone on the <x> <y> coordinate of the game board.
    - Assume board size of 10 x 10.
    - If a player uses this command when it is not his/her turn, you may simply ignore, or you may print out a message *"not your turn"*.
    - If the game is already over (win/loss already decided), ignore.
  - If the user makes an invalid move
    - (e.g. <x> or <y> out-of-bound, or there exist a stone at <x> <y> already, etc.),
    - you must detect that and print out a message *"invalid move"* on the client.
  - If it is a valid move, all clients will print out a new game board (with latest move), and each client will individually check whether you **win or lose**.
  - If the game is already over, ignore.
  - While playing the game, put 10 second **timeout** timer for every move.
    - That is, if a player does not play his/her turn within 10 sec, that player **loses**.
    - If the game is already over, ignore.

Command details

- **\gg**
  - Gives up the game (you lose). If the game is already over, ignore.

- **\exit**
  - Identical to pressing '`Ctrl-C`' while playing the game.
  - If the game is not over yet, you lose, opponent wins.
  - You notify the peer that I'm exitting, and quit(terminate) the program.
    - That means, a client should send a message to the peer before terminating.

- If you win, your client will print *"you win"* message, the peer will print *"you lose"* message.
  - Think about { win/lose, timeout, \gg, \exit, '`Ctrl-C`' } cases.

Other additional requirements:

- Your program must run on our class Linux server at nsl2.cau.ac.kr.
- Running your client on nsl5.cau.ac.kr and server on nsl2.cau.ac.kr should work without ANY code modification.

- Make sure that you use your own designated port number for the server socket.
- For the client sockets, you should not set the port number, or must use null (0) port number which will let the OS assign a random port number to your socket.

- You should close all sockets when exiting. You should be able to restart any of your programs immediately after exiting/terminating.

- Your code should be easily readable and include sufficient comments for easy understanding.
  - Your code must include your name and student ID at the beginning of the code.
  - Your code must be properly indented and formatted.
  - Your code should not include any Korean characters. Write your name in English.

- Not satisfying any of above requirements will result in serious point deduction.

What and how to submit

- You must submit softcopy of **P2POmokClient.go** and **P2POmokServer.go**.
  - Client should take one command line argument, <nickname>.
    - Ex> `$ go run P2POmokClient.go ironman`
    - Ex> `$ go run P2POmokClient.go superwoman`
  - Server should take no command line argument.
  - Your program should work even if there are 3 or 4 or 5 peers running.

- Here is the instruction on how to submit the softcopy files:
  - Login to your server account at nsl2.cau.ac.kr.
  - In your home directory, create a directory "submit_net/submit_<student ID>_hw5" (ex> "/student/20229999/submit_net/submit_20229999_hw5")
  - Put your file and only that in that directory.
    - Do not put any code that does not work! You'll get negative points for that.

Grading criteria

- You get 5 points
  - if all your programs work correctly, AND
  - if you meet all above requirements, AND
  - if your code handles all the exceptional cases that might occur, AND
  - if your code is concise, clear, well-formatted, and good looking.
  - Otherwise, partial deduction may apply.
- You may get optional extra credit of up to 2 points
  - if you do the optional extra credit tasks.
- No delayed submissions are accepted.
- Copying other student's work will result in negative points.
- Code that does not compile or run will result in negative points.

[Optional] Extra credit task

- Do the same thing as above also in **JAVA** (up to 1 point)
  - For submission, your file names should be same as Golang counterpart except the file extension (.go ➔ .java). Your JAVA programs should compile with javac.
  - If you do this, not only your JAVA client should work with JAVA server, but your Go programs should also work with your JAVA programs. That is, you should be able to mix and match JAVA and Go programs for server and client, in any combination.
  - Do not submit if it does not work. You will get negative points for that.

- No-extra-points, just for fun!
  - Implement '\please' command which retracts a move during the game
    - (maybe, the opponent must agree)

- Implementing a **GUI** for this program! (up to 1 point)
  - if you do this, add a README file to explain how to run.