Final project for: Windows OS - cyber viewpoint

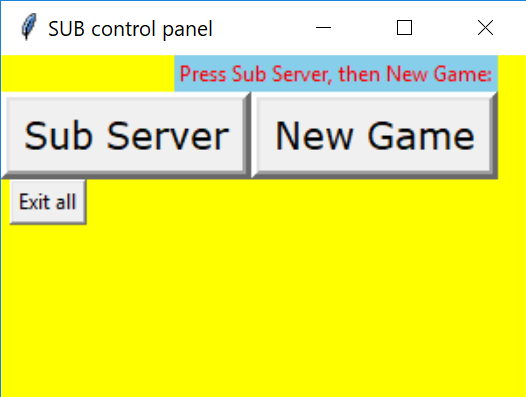
Teacher: Jacob Shutzman ID: 05609316

Project subject: **Submarines and chattings**

This is an interactive game between 2 playes over the internet. It could be enhanced easily to have multiple players. In addition to playing the game, the opponents can chat with each other during the game, as if they are in a chat room (teasing each other maybe...)

The game: This is a board game of submarines. Each player has a board of 10x10 squares. The board could have different pixel size by changing one global parameter: PIXEL\_SIZE (preferably a number divisable by 10)

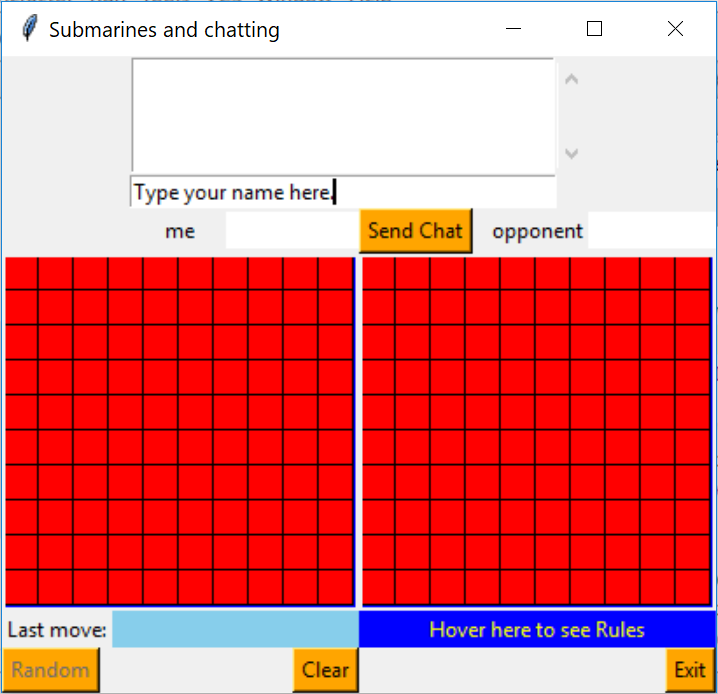
The project includes a small GUI to operate the game, control panel , if you will. It has a button to start the server (Sub Server) and a button to start a game (New Game). Here is how it looks:



The button ‘Exit all’ is used to shut down the server and close this control panel. So, normally if played on one computer, you’d click once on the ‘Sub Server’ and twice on ‘New Game’.

Here is how the game looks like when we first run the program (New Game).

[ picture of the initial board ]



On the left hand side where it says 'me', is the board of the player seeing it. On the right hand side is the opponent's board (where we cannot see any submarine during the game)

Buttons and features explanation:

**Random** - Randomly creates the submarines for 'me', and sends them  
 over the network to my opponent. The game has one  
 submarine size 4 squares, 2 size 3, 3 size 2 and 4 size 1. It  
 makes sure they do not border each other. The button turns  
 white (from orange) after it has been pressed. This button is  
 disabled initially. It becomes active, after you send your name  
 via the 'Send Chat' button, or type your name and hit <enter>   
 and another player is logged on.

**Send Chat** - If we want to chat, we'll write the message at the box above  
 the button and press this. You may also hit <enter>  
 instead of using the button.

The messages show on the list box above where it's written.   
 we can scroll on the list box to see more messages. Both   
 ‘Random’ and ‘Send Chat’ have tooltip (when the mouse   
 hovers over them, there is explanation of what they do.

**Clear** - Clears out all of the submarines generated by Random. It   
 also cleans all the misses, hits, drowned subs from the   
 opponent’s board. If one player clears it is recommended that  
 the other does too, so they can start a new game by both clicking  
 Random.

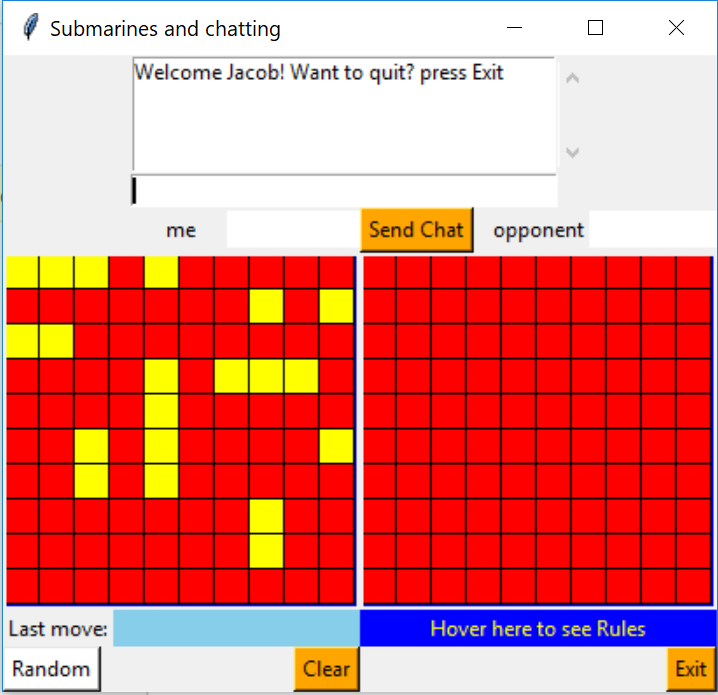
**Exit** - If we want to completely exit the game. Can be used   
 by either player independently. It closes the socket, closes the   
 GUI and exits back to the OS. It indicates to the server to do  
 some cleanups as well, but more more importantly, If a player  
 exits, the server remains running and another player. We later  
 discuss more about the handling of the server.

**Stats** - This button only shows up for the winner, when the game ends.  
 Pressing it will present the number of guesses and accuracy on  
 top of the opponent’s board, (blue background and white   
 foreground, and for fun, the text changes color if we hover over it  
 with the mouse.

**Hover here to see rules** - If you hover with the mouse over this blue  
 label you can see a brief explanation on how to play. We’ll see   
 a smaple screen later.

Here is how the board looks like after Random has been pressed. Of course after every press on Random, it would have a different arrangement of the subs. In order to press Random, we must enter our name (otherwise the button is disabled).

[ board with generated subs, after pressing ‘Random’]



Before starting the game, it would be nice for each player to introduce themselves, and at least enter their names in the chat feature. They do not need to chat if they are not interested in chatting, however they must send at least one message via the 'Send Chat' in order to be able to play (more on that in the following).

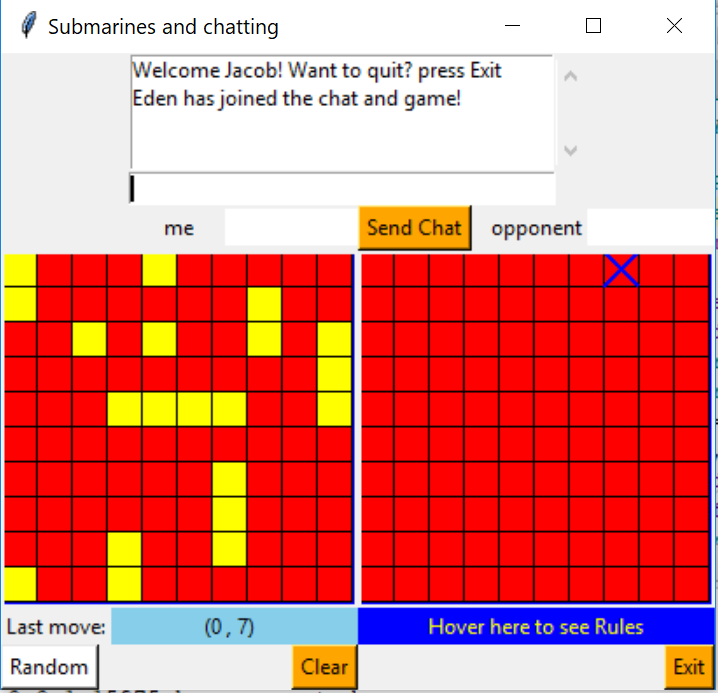
Starting the game: Each player needs to press their Random button, so the system generates all of their submarines and sends it to the opponet's side (hidden). This could be encrypted, but since it is only a fun game it would be an overkill (although a good exercise as a project). Before the player presses Random, he or she is forced to send one message (name) to the other side, as we have mentioned before. The screen above is already after sending the name and having another player loggoed on.

**Purpose of the game:** Trying to guess and hit the opponent's submarines. The player who hits all of them first, wins.

The interface is very simple. By clicking on the opponent's board, at any square, we may get one of 4 outcomes. If we miss, we get a white circle and hear a 'whiff' sound. If we hit a part (a square inside a sub) we'll get an X inside that square and a 'shot' sound. If our click 'drowned' the sub (all its squares are hit), we have it painted in green and we hear a longer chearful sound. If our hit was the final blow to the whole fleet of submarines of our opponent, we get a 30 second standing ovation music and a 'flashing' sign of 'WINNER'. I chose not to humiliate the loser so currently he or she will have just a short (5 seconds) sound of losing with indication of who won and who lost. They’ll also see that all their subs were drowned, of course. The ‘Stats’ button will show up for the winner only.

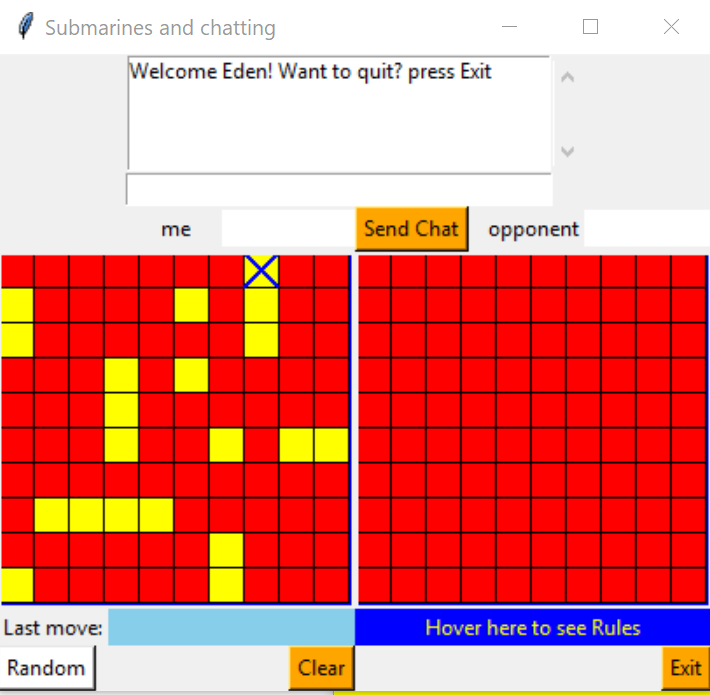
Here are the different views of 'hits' (X in the square), drowned (green painted sub) and game won.

[After hitting a submarine-The view of the player who hit (pressed [0,7]) ]

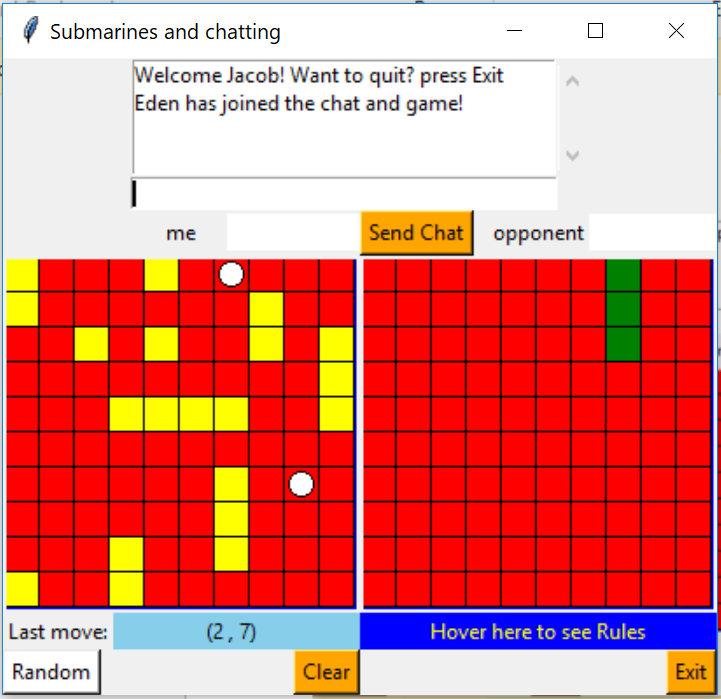


Note on the skyblue background the last move displayed.

[Here is how the board of the opponent looks after hitting her [0,7] ]

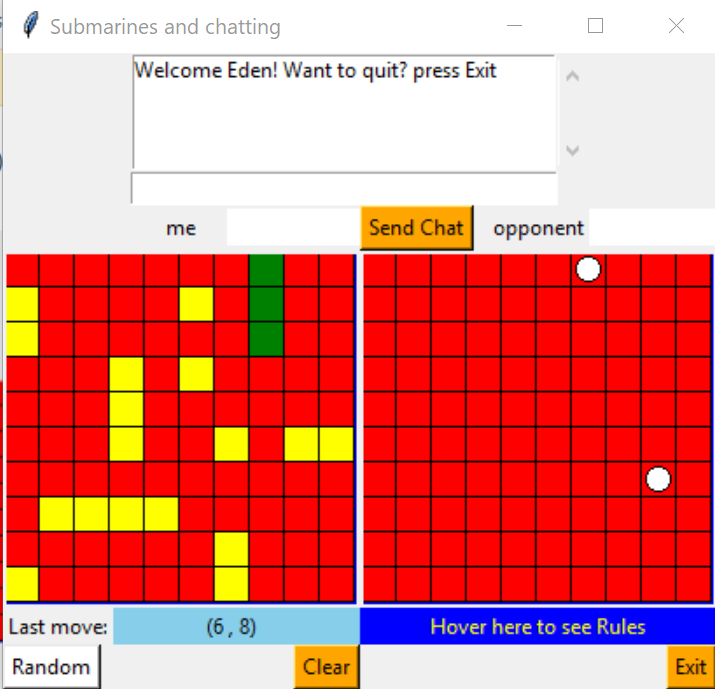


[My board after me drowning a sub in (0,7) thru (2,7) ]



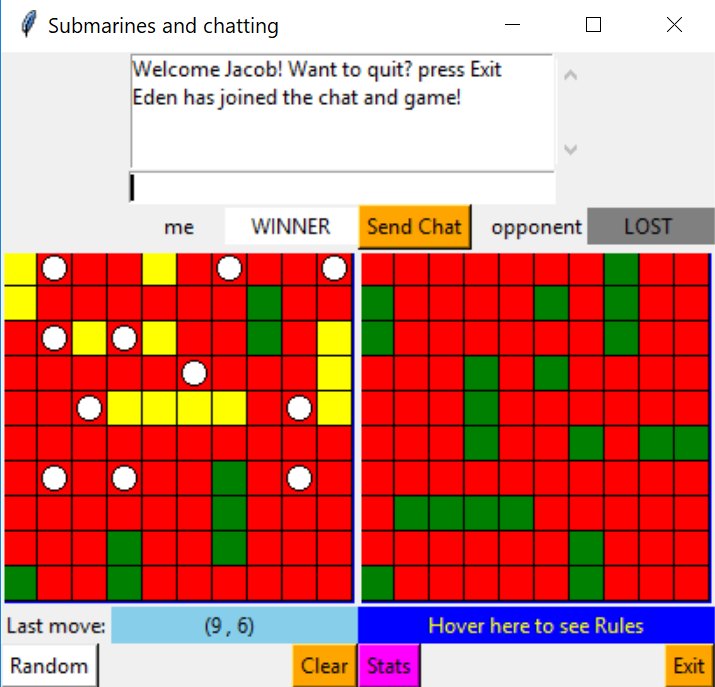
The two white cicrles on the ‘me’ board are the misses of my opponent.

[Opponent’s board after her sub in (0,7) thru (2,7) has drowned]



Note that here the opponent’s misses show up on her ‘opponent’ board.

[ This is how the winner’s board looks like. WINNER is flashing white/red]



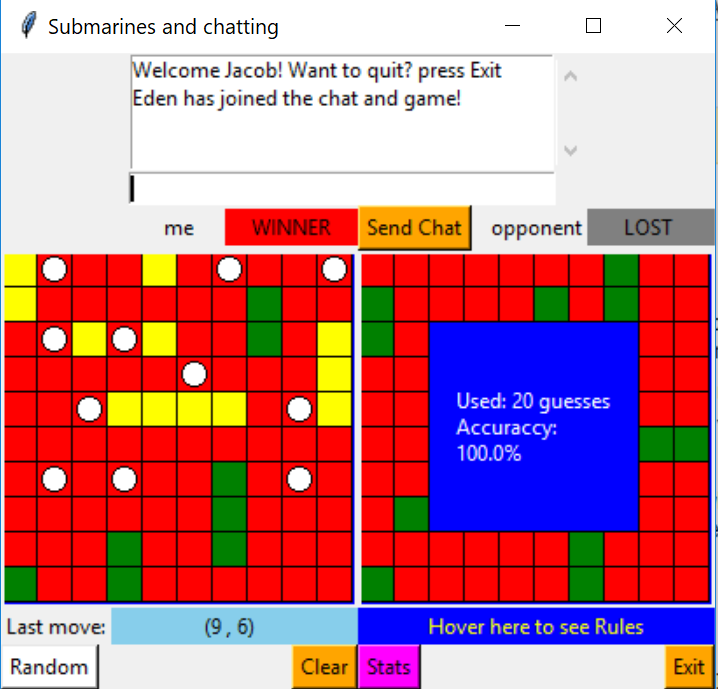
The white circles in the above board are missed shots of the loser . My last move was (9,6). Note that many of my subs (in yellow) are still alive.

[The loser’s board after the above game was over]



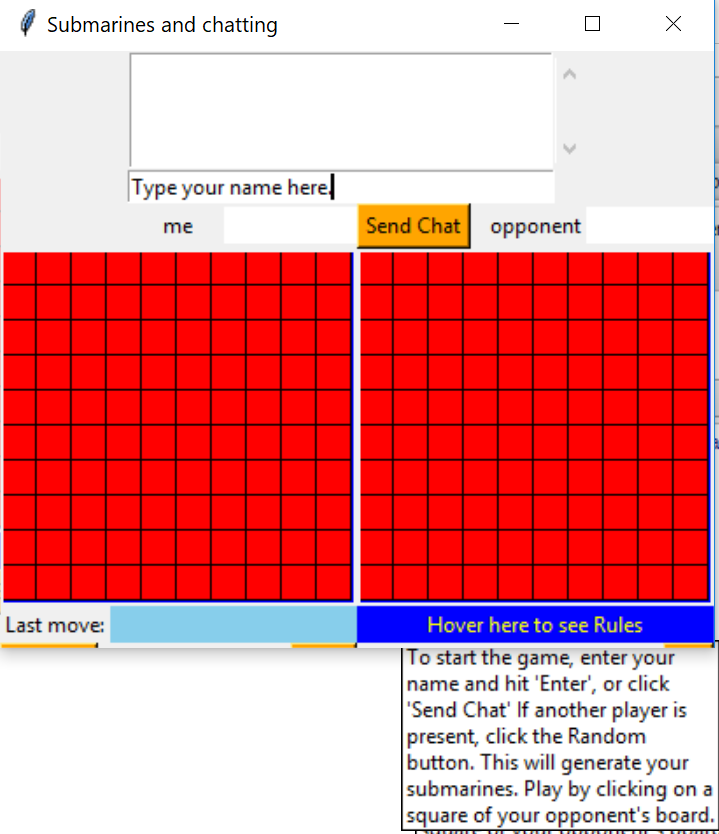
Note the Lost/Winner signs. Here, the ‘me’ is the loser and none of the subs is alive (all green). The opponent (Eden) drowned 4 of my subs only.

To see my stats after the game, notice the magenta background button that says: Stats (on the winner’s screen). Pressing it we get:



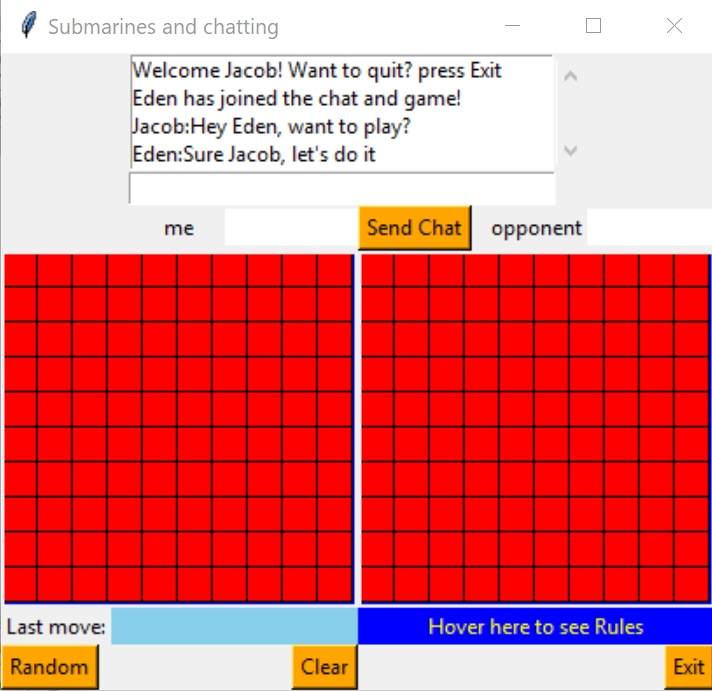
Because I played both players on the same computer, I had no misses and hit with 20 shots all of Eden’s submarines. Otherwise the accuracy would be the percentage of good shots (20 out of however many taken).

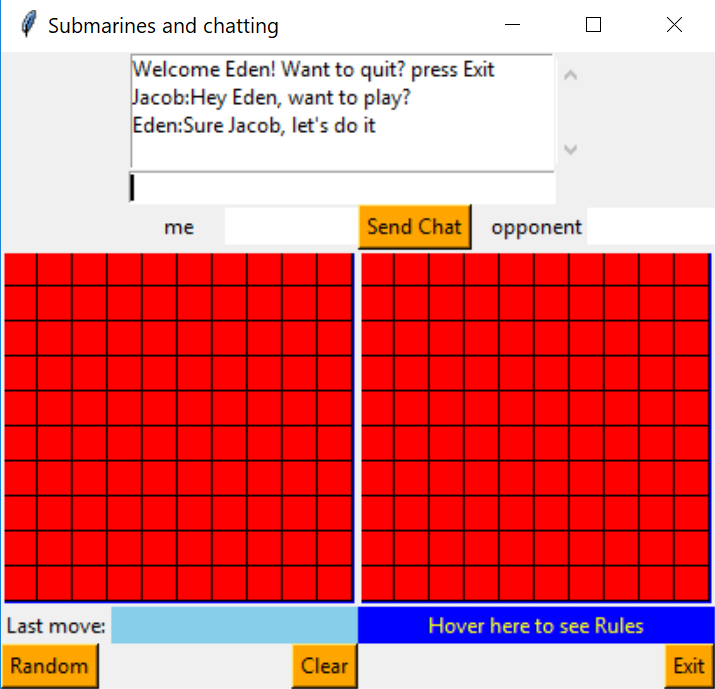
[Here is how the screen looks when we hover over the rules]



The chatting portion of the application keeps the party who is still in, informed if the other player has quit. Any player can quit the game at any time, and anyone new (who knows the ip/port) can join.

Here is how a short chat would lok like on both sides:





This chat obviously took place before starting the game.

Technical explanation / developer guide

The game is written in Python, using tkinter for the GUI and socket module for the communication. The submarine fleet is defined as a dictionary that is being transferred as plain text with json dumps and loads. A submarine is a list of lists, for example: [[1,1,False],[1,2,Fale]] This is a submarine of 2 horizontal squares located on the second row, columns 1,2. False means it has not been hit yet. In class Board the instance variable self.submarine is a dictionary, whoes key is a 2 digit number, the first is the sub’s size, and the second its sequence number. First in the sequence is the 4-square sub (seq=0), then the 2 3-squares subs, then the 3 2-square subs and then the 4 1-square subs. So it may look like this (only the first 2 subs):

{ 40: [[1,1,False],[1,2,False],[1,3,False],[1,4,False]],  
 31:[[0,3,False],[1,3,False],[0,4,False]],....... }

There are 2 main modules, one is a server (subserver.py) that has to be run first on one computer, and the two players are clients (subs.py) that can be run on separate computers (of course all 3 can run on the same machine with 127.0.0.1). There is also another GUI module: guiclientserver.py, which can be used to launch server and clients.

The server allows a player to disconnect when they press Exit, and then to accept a new player, even at a middle of a game, or before the game starts after the subs are generated or even before. This requires ending the threads, and telling the server that a client is quitting, so the shutdown is done gracefully. The global variable ‘players’ is updated as the number of players logged on to the server. The server sends it to the client programs, so they can know whether or not to allow generating the subs (only when 2 players are loggoed on). If only one player is loggoed on, the client program leaves the Random button disabled.

The clients start by providing their IP and PORT (port can be left empty) the default port (for sending) in the client, is the server's port. Currently the client IP’s are ‘Home’ (127.0.0.1) to enable play on the same machine. The appropriate part of the code needs to be uncommented, for receiving IP and PORT from the input line.

The board is defined as a class, so we create 2 Board objects, one for the player (me) and another for the other player (opponent).

Usage of threads - Those are used in the client for both the purpose of making the winner sound (which is 30 seconds long), so the screen doesn't stall and the music is only in the background.

It is also used in the communication part, because of the tkinter's mainloop feature. This is one way of overcoming an infinite loop within an infinite loop. They are also used in the server where each client is a thread, and the client handling during the game/chats is a thread as well.

There is one more class called CreateToolTip which is capable of creating a tool tip for any widget (it is only used for ‘Random’ , ‘Send Chat’ and the game’s rules )

The communication protocol

For every message transferred from a client to server and vice versa, there is a 4 character prefix. The options are: ‘chat’, ‘rand’ or ‘move’. Accordingly the server analyzes the type. If it is a ‘chat’, it sends the message to both (or all) players. If it is a ‘move’ or ‘rand’, it sends the message to the player whose turn is coming up. ‘rand’ is after one player pressed Random and his board generated the subs, and now he sends their locations to the other player. ‘move’ is after clicking a move on the opponent’s board. The server (subserver.py) uses the order in which the players logged in, with their corrsponding sockets, to determine whose move it is (the opposite of the one who has sent the last message). In order for preventing abuse of the game, when a chat request is sent to the server, the server sends it back with the number of players loggoed on currently (5th byte, right after the prefix ‘chat’). This number allows the client program to turn the ‘Random’ button on or off (on when more than one player is present, off otherwise).

The GUI (subs.py) ‘receive’ function interprets the message. If it’s a ‘chat’, it inserts it to the ListBox, updates the number of players (and accordingly the state of the Random button). If it is ‘rand’, it adds the submarines sent to the opponent’s board, and if it is ‘move’, it plays it on the ‘me’ board.

The server also keeps the players sockets in a dictionary, and uses it for ‘broadcasting’ messages to all (the two players), as a chat room style (as we saw before):

player1\_name: Hello there  
player2\_name: Hi

Potential abuse scenario and how we overcome it

2 players log in, then both leave. One gets in and sends her name to the chat. Random button is still disabled (only one player present). The second logs in and gives her name. Now Random is enabled for both. The second now leaves, the Random button becomes disabled again (for player 1 who remains). The second player logs in again, gives her name, Random is enabled for both. The first player presses Random, the second then leaves, the first presses Clear, Random is no longer enabled for the first, because she remained the only player. If the second player logs in again and gives her name, Random becomes enabled for both.

That leads us to the conclusion that if sometime in a middle of a game, a player quits, the one remaining can wait for an opponent. When the opponent logs in, the best practice for the one waiting, is to hit the Clear button, and only then, to hit the Random button to start a new game.

More about the server

The server ‘listens’ to up to 5 clients at the same time (although only 2 are needed for the game). It uses a thread to run the acceptance of new players (endless loop). The server has a number of dictionaries to store client name, address (ip+port) and ‘order’ which keeps the sequence of who logged on first and who did second. All 3 dictionaries use the socket object as the key. The 4th dictionary is the order\_rev (reverse: get the socket for the keys 1 or 2). The acceptance loop stores addressess, order and order\_rev (name is not know yet). For simplicity all dictionaries are defined globally.

The acceptance loop starts another thread of ‘handle\_client’. This function welcomes the new player and announces to all other players (1 at most right now), that someone new had arrived. It then runs an endless loop of receiving messages. If a message is a ‘chat’, it broadcasts it to the two players, otherwise (’move’ or ‘rand’), it sends it to the opponent of the one it was received from. When a player quits it denotes a ‘chat’ with the string: {quit}. If that was received, the server does some cleanups and disconnects the socket to that client, and announces to all who has quit. The cleanups include deleting the dictionary entries and resetting the players to 0. The acceptance of new player never ends, even if both players quit, so the cleanup is necessary for new players that may log on. It can only be ended by the control GUI guiclientserver.py with the button ‘Exit all’. This control program also uses threads to activate the server and clients. It imports the other modules (subserver, subs) and activates their ‘main’ method to run them. It also use the method shut\_down\_server of the subserver, when exiting, to shut down the server.

The project is written in Python 3.x

Here are the files that make up the project:

guiclientserver.py

subserver.py

subs.py

sound files: hit.wav, hitmiss.wav, drown.wav, winner1.m4a, losing.m4a

project\_book.docx (this document)

Python modules imported

tkinter , random, json, math, playsound, threading, socket

Possible enhancements

1. Forcing the alternating moves. Currently the same player can make multiple moves, before the opponent makes a move.
2. Tightening the control to make it full proof, so no one can do anything illegal, by adding error messages box.
3. Checking for a legal name with a regex.
4. Use encryption/decyption for the chat, so no one can evesdrop on it.
5. Allowing the players choos their own locations for the subs. Random was created to save time, but on top of it we can allow manual changes.