Tic Tac Toe Summary Document

Brief Description of Choices Made

In brief: the server boots up first, opens a socket, and listens for connections. The client then boots up using as command line arguments the server's IP Address and Port Number. **The Client makes all the choices in my application** (X/O & first/second assignment, and play again choice)**.** If they play again the connection remains open, the board is reset, and game parameters are again set by the client. Otherwise the connection is closed by the client. The server will remain listening for connections until ctrl-C closes the server file.

When facing design decisions I tried to go for simplicity, to hopefully aid in cross-compatibility for HW5.

My Tic Tac Toe game represents the state of the board through a string which gets passed from player to player, and to make a move in the game is to manipulate the string. The string is 10 characters long, and represents the 9 positions on a tic tac toe board just like they are lined up visually on a keypad.

```
   |   |
-----------
   |   |
-----------
   |   |
```

For example: an empty board is represented as: "GNNNNNNNNN"

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

This position above would be: "GONXNNNXNN"

Files

ttt.py
A driver file that can launch either server or client mode.
To run as the server: python3 ./ttt.py -s
As client: python3 ./ttt.py -c <ip_address_server> <port_num>
Server with autoplay: python3 ./ttt.py -sA
Client with autoplay: python3 ./ttt.py -cA <ip_address_server> <port_num>

tictactoe.py
Client file.

tictactoed.py
Server file.

tictactoeA.py
Client with autoplay.

tictactoedA.py
Server with autoplay

gamefunctions.py
Hosts helper functions utilized by both client and server files.

README.md

Helper Functions

checkBoard(position)

A simple checker function that makes sure our game-state string is valid. **Returns 0 if valid, -1 if not.**

printBoard(position)

Displays current state of board on command line for players.

makeMove(position, player_char)

player_char is either 'X' or 'O'. makeMove takes an integer 1-9, and then **returns str new_position.** new_position is a new state of the game. I try to have this function do some quality checking for user input. If user selects a square that's taken, for instance, or a number that's not 1-9, they get another chance to give a valid answer.

checkWin(position)

This takes a board state and simply checks for a winning position. **Returns 1 if X wins, 0 if O wins, 2 if its a tie (so all moves have been made, board is full), otherwise -1.**

GameSummary(position)

This function calls checkWin on position and then prints and appropriate message.

moveFirst(socket, id, Current_board)
moveSecond(socket, id, Current_board)

id is 'X' or 'O', Current_board is the string of game state, which will always be a fresh board for this function, socket is the connection socket.

These two functions can work for both client and server. Depending on who is going first or second, they get called, and the gameplay happens within a while loop within them.

moveFirst makes a move, checks it for a win, sends the state over, and then receives a new state back from the opponent all within a while loop that gets broken out of when there's a winner.

moveSecond is similar except it receives a move first, makes a move, checks for a win, then sends said move within a while loop.

Client-Only:

setUpGame(client, server, first)

Takes command line input from client, determining who's X and O, and who will go first.
Returns client('X' or 'O'), server('X' or 'O'), and first('C' or 'S')

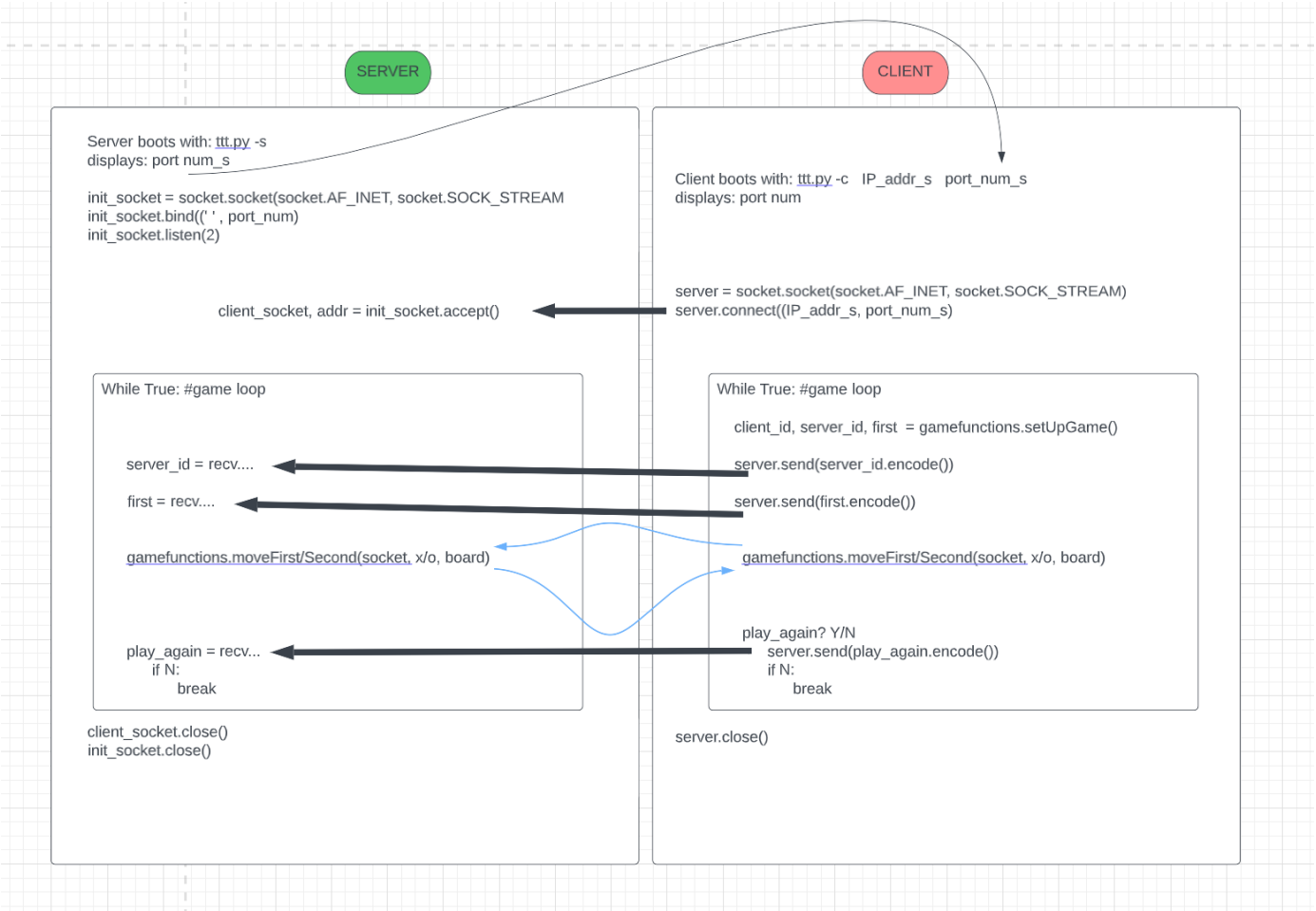
checkToPlayAgain()

Checks if client wants to play again. **Returns play_again, either 'Y' or 'N'.**

Extensions

I made an automatic player. It does not play optimally, but simply blocks its opponent when the opponent is one square away from winning. Otherwise it picks a random available square. If you can 'fork' it, and set up a scenario where there are two potential wins, you can beat it. I found online steps to build an optimal player, but decided against it just for time management.

Diagram



Made with http://www.lucidchart.com

This diagram is not comprehensive, but I'm making it as an aide to visualizing my program, and specifically where the sockets do things.
The black arrows represent the client establishing a connection socket, and then passing information about decisions the client makes to the server.
The game flow happens in the blue arrows, within the functions moveFirst & moveSecond.