COMPUTER NETWORKS

Professor: Yanni Liu

Course: CSE 310. 01

PROJECT DOCUMENTATION
TIC TAC TOE GAME

Group Number: 27

KRISHNA ARJUN SARAVANAN

RAHUL S AGASTHYA

SAMUEL MCKAY

TABLE OF CONTENTS

INTRODUCTION

The course "Computer Networking" introduces the overview of computer networks and the Internet. Discussed in depth are the concepts of End Systems, Access Networks and, Client and Servers. The course is focused on the TCP/IP stack, and other concepts such as Connection Oriented and Connectionless services, Circuit and Packet Switching and, most importantly Socket Programming. The layers of the Internet Protocol Viz. Application, Transport, Network and Link layers were discussed in detail.

This project the "Tic Tac Toe" game is a requirement for this course and is the crux of the final grade. This project focuses on Socket Programing and the Client-Server interaction. The program implements a simple transfer of message between the Server and the Client. The client is connected to the server using a TCP.

Our project successfully accepts the following commands:
`login <user_id>`. Accepts a unique user_id and the game identifies the player through that name. The initial status of the player is set to "Available."

`help`. Displays the help menu of the game.

`place <index 1-9>`. Accepts an index from 1 to 9 and places the 'O' or 'X' in the respective cell.

`exit`. The player exits the game and is disconnected from the server.

The program also handles for error conditions in each of the above conditions.

PROJECT OVERVIEW

The main objective of this project is to "Design and implement an online modified version of the game Tic Tac Toe using Internet Domain sockets."

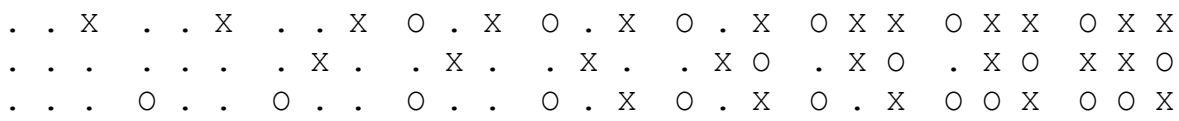
The game is played between two players. The game board is a 3 × 3 grid comprising 9 cells. The player who logs in first shall get the shape 'X' and gets to move first, while the other player uses 'O'.

The game condition is if either player gets the following, he/she loses:

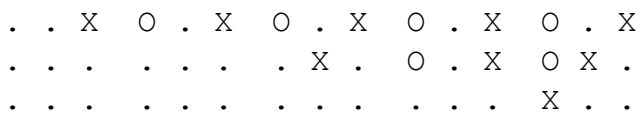
- 3 in a row
- 3 in a column
- Along the either diagonals.

The objective of any player is to force the opponent to reach either of the above-mentioned conditions for the player to win.

An example is as follows:



This is a draw game.



Here, 'O' wins the game.

PROJECT SPECIFICATIONS

Project Requirements:

The project required a client server model. Each player is a client, while the game runs on the server.

The client sends the commands to the server and receives the relevant response either as updated game board, help menus, error and other messages. Based on the content of these messages, the client decides the future moves which shall be permitted.

Clients connected to a particular game must get both generic messages like the status of the game board and custom messages like error messages or help messages, through Multiplexing.

Enabling the program to handle multiple games at any point in time, though Multi-Threading.

Program Requirements:

A. Pre-Requisite Knowledge:

While running this program, there is no pre-requisite knowledge required. However, basic knowledge of terminal is advised.

To run the server, the user must type (The Python Software Foundation):

```
python Server.py
```

To run the client, the user must type:

```
python Client.py
```

B. Hardware Requirements:

The program runs only on the terminal or a windows shell, as the program uses `select()`, and the use of this function is limited to terminal or shell. No other computer hardware is required, apart from the keyboard and a monitor.

C. Software Requirements:

The computer must have Python 3.0 or higher. This is available for free online.

ALGORITHMS AND APPROACH

Socket Programming in Python: (Tutorials Point (I) Pvt. Ltd.)

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, etc. (Here, we use TCP). The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest (Tutorials Point (I) Pvt. Ltd.).

A. Vocabulary of the sockets:

Term	Description
Domain	The family of protocols that is used as the transport medium.
Type	The type of communication between the two endpoints. Here, we use `SOCK_STREAM`.
Protocol	This is used to identify a variant of a protocol within a domain and type. Here, it is 0.
Hostname	The identifier of the network interface. Here, it is 'localhost.'
Port	Each server listens for clients calling on one or more ports. We use port '8080.'

To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax

```
sock = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters –

- `socket_family`: This is either `AF_UNIX` or `AF_INET`.
- `socket_type`: This is either `SOCK_STREAM` or `SOCK_DGRAM`.
- `protocol`: This is usually left out, defaulting to `0`.

B. Server Socket Methods

Method	Description
`s.bind()`	This method binds address (hostname, port number pair) to socket.
`s.listen()`	This method sets up and start TCP listener.
`s.accept()`	This passively accept TCP client connection, waiting until connection arrives (blocking).

C. Client Socket Methods

Method	Description
`s.connect()`	This method actively initiates TCP server connection.

D. Other Socket Methods

Method	Description
`s.recv()`	Receives TCP message. Used in project.
`s.send()`	Transmits TCP message. Used in project.
`s.recvfrom()`	Receives UDP message.
`s.sendto()`	Transmits UDP message.
`s.close()`	Closes socket.
`socket.gethostname()`	Returns hostname.

Basic Algorithm:

The `Server.py` runs in the background, while the clients run the `Client.py` files on their respective machines.

As a player logs in, a new `Player` object is created and is added to the players list in the `Server.py` file.

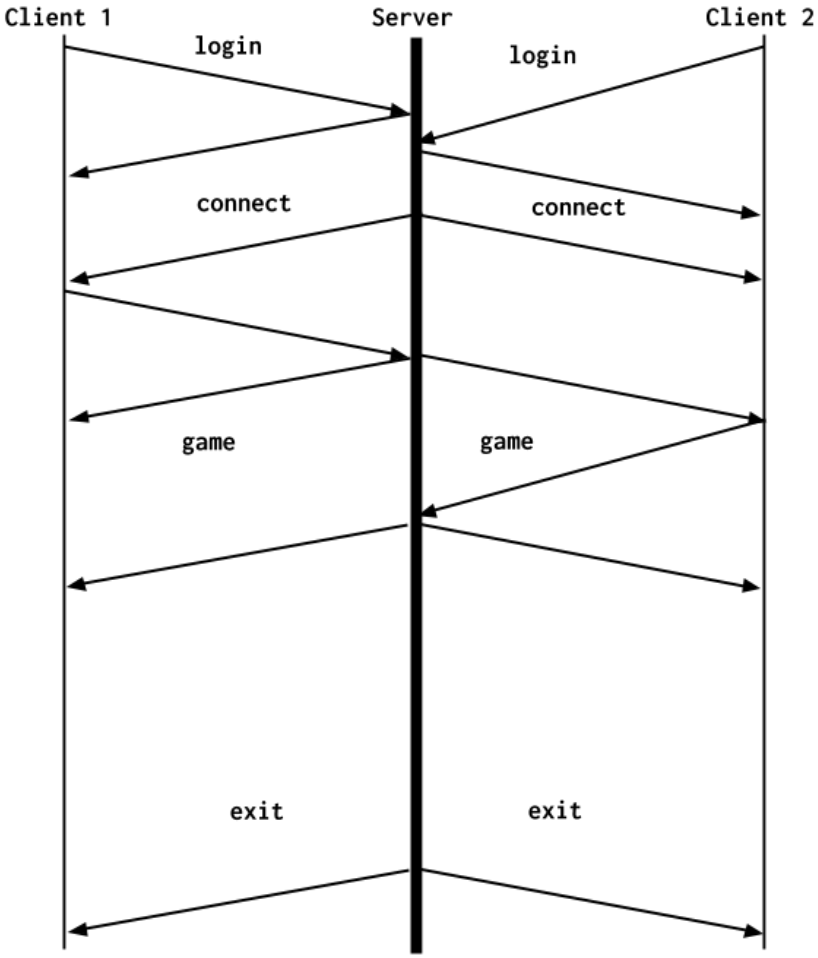
After two players log in, the `Server.py` sends out the creates a new game and sends the initial game board to both the players. Also, the player who logged in first gets 'X' and gets to go first.

Each command entered by the client is relayed to the `Server.py` using the `send()` method, while the sockets reads this using the `recv()` method. Once, receiving the command, the `Server.py` responds to the request and sends out a message, which is read by the `Client.py`, and the game goes on.

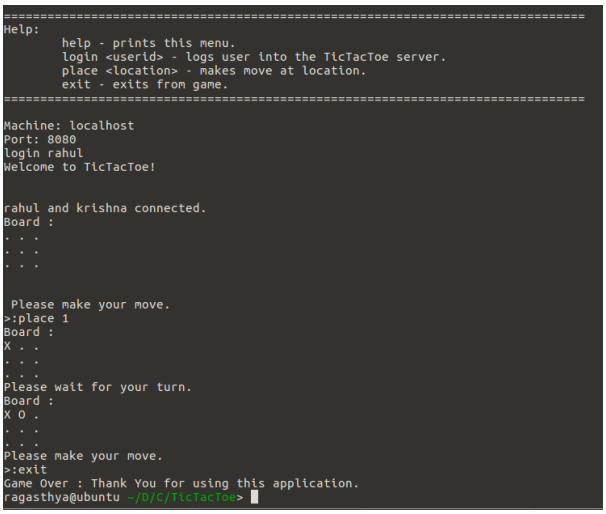
If either of the players reaches the game condition, then appropriate messages are sent to the winner and loser and the client exits. This is true if the game is draw.

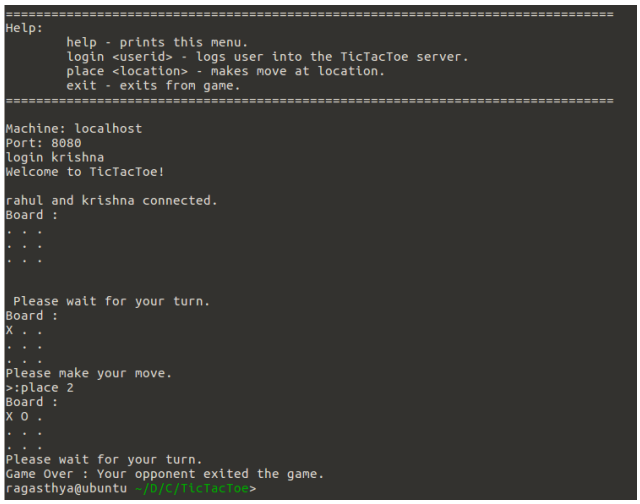
The subsequent sections of this documentation show a sample run and the screenshots are provided.

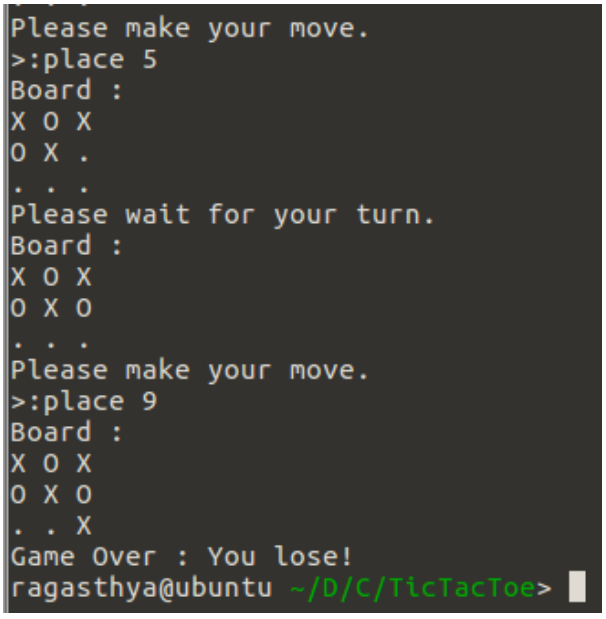
Sample Run:

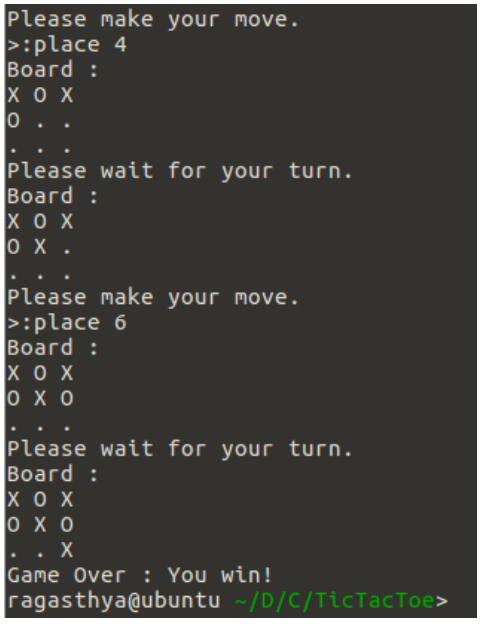

Client 1
Server
Client 2
login
login
connect
connect
game
game
exit
exit

Screenshots:









Future Expansion

In the future, this project can be further developed to have a user sign up and user authentication. The program can further be advanced to use a graphical user interface rather than the boring command line or terminal interface. This can also enable the evolution of the project into a mobile application.

In terms of networking, the project can be further developed to include observing another game, and to allow the players to communicate with each other through a chat window.

References

The Python Software Foundation. *Python Documentation*. 26 March 2017. <docs.python.org>.

Tutorials Point (I) Pvt. Ltd. *Tutorialspoint.com*. 24 June 2014. <www.tutorialspoint.com/python>.