PROJ01: Tic Tac Oh No!

Kevin Scrivnor – Version 1.0s21

Table of Contents

> `return ('t'&'r'&'u'&'e') == ('f'&'a'&'l'&'s'&'e');`
>
> — *Steven Rostedt*
> *Maintainer of ftrace*

Due Fri Mar 5

changelog

2/19/2021 v1.0 initial release

Overview

Goal

Your goal is to implement your tic tac toe protocol so that two clients may play a game of Tic Tac Toe over the Internet. A Tic Tac Toe engine is given below to save you time. This document also contains a sample threaded server and a client implementation.

Requirements

- Protocol documentation.
- Implementation of a client-server architecture for multiplayer tic tac toe.
- It must be done with Python 3.
- The transport layer must be TCP.
- The provided engine must be used, though it may be lightly modified to fit your needs.

Grading

Points	Area
10	Updated Protocol Documentation
70	Correctness and Code Quality
20	Testing

- Correctness and Code Quality
  - 70 - Fully working implementation, commented code, senior level programming
  - 65 - Fully working implementation, some comments, senior level programming
  - 50 - Some features missing
  - 35 - Many features missing
  - 0 - Code does not execute
- Updating Protocol Documentation
  - 10 - Accurately and reasonably describes the workings of your protocol (includes examples)
  - 9 - Accurately describes the workings of your protocol (includes examples)
  - 8 - Accurately describes the workings of your protocol but missing some descriptions.
  - 7 - Does not necessarily reflect the protocol
  - 0 - No document
- Testing (varies)
  - 20 - Tested/handled the fringe cases that the protocol may enter
  - 18 - Handled the fringe cases but not tested during testing
  - 15 - Tested fringe cases but not handled by the code
  - 5 - Missing test cases/not handling odd cases
  - 0 - No testing

Cheating

Any academic dishonetly results in an automatic fail (0/100 points) added to the exam portion of your grade. Meaning, if you receive a 100 on the midterm and 100 on the final, your exam total will be: 200/300 = 66%.

Reading

- Simple TCP Example (https://pymotw.com/2/socket/tcp.html)
- Python Tutorial (https://www.w3schools.com/python/default.asp)

Example Code

Tic Tac Toe Implementation

Below is a Tic Tac Toe engine written in python. You can run this just to see how the various functions work.

PYTHON

```python
class TicTacToeEngine:
    def __init__(self):
        # board is just a list of dashes for blanks. X and O's will fill it eventually.
        self.board = ['-','-','-','-','-','-','-','-','-']
        # is it x's turn?
        self.x_turn = True
        # how many successful turns have we played?
        self.turns = 0


    def restart(self):
        self.board = ['-','-','-','-','-','-','-','-','-']
        self.x_turn = True
        self.turns = 0


    def display_board(self):
        j = 0
        for i in range(0,9): # for (i = 0; i < 9; i++)
            # print without a new line
            print(self.board[i], end=' ')
            j += 1
            # add a new line every 3 board spaces
            if j % 3 == 0:
                print('')


    def is_game_over(self):
        # winning combos in tic tac toe
        winning_combos = [  (0,1,2),(3,4,5),(6,7,8),
                            (0,3,6),(1,4,7),(2,5,8),
                            (0,4,8),(2,4,6)]

        # for each of the winning combos
        for combo in winning_combos:
            # assume the first piece is a winner
            winner = self.board[combo[0]]
            # if it is not blank
            if winner == 'X' or winner == 'O':
                # and if the next two on the board are the same as the first one
                if winner == self.board[combo[1]] and winner == self.board[combo[2]]:
                    # that piece has won
                    return winner

        # no winning combos and the board is full.
        if self.turns == 9:
            return 'T'

        # not done yet
        return '-'


    def make_move(self, pos):
        # make a move if it is valid (between 0 and 8 inclusive)
        # increase number of turns by 1
        # invert the x_turn boolean
        if self.is_move_valid(pos):
            if self.x_turn:
                self.board[pos] = 'X'
            else:
                self.board[pos] = 'O'
            self.turns += 1
            self.x_turn = not self.x_turn
```

```
            return True
        return False

    def is_move_valid(self, pos):
        # make sure it is on the board and no one has already plkayed there!
        return (pos >= 0 and pos <= 8 and self.board[pos] == '-')


ttte = TicTacToeEngine()
ttte.display_board()
print(ttte.is_game_over())

for i in range(0,9):
    print('='*40)
    ttte.make_move(i)
    ttte.display_board()
    winner = ttte.is_game_over()
    if winner != '-':
        print("Winner: " + winner)
        break

if winner == '-':
    print("Tie.")
```

Simple Server Example

PYTHON

```python
# server.py - a simple threaded server

import argparse, socket, logging, threading
# Comment out the line below to not print the INFO messages
logging.basicConfig(level=logging.INFO)

class ClientThread(threading.Thread):
    def __init__(self, address, socket, thread_cond):
        threading.Thread.__init__(self)
        self.csock = socket
        self.address = address
        self.thread_cond = thread_cond

        logging.info('New thread!')


    def run(self):
        # send a message
        self.csock.sendall(b'Hello client!\n')

        # get a message
        msg = self.recv_until(self.csock, b"\n").decode('utf-8')
        logging.info("Message: " + msg)

        # disconnect
        self.csock.close()
        logging.info('Disconnect client.')


    def recv_until(self, sock, suffix):
        """Receive bytes over socket `sock` until we receive the `suffix`."""
        message = sock.recv(1024)
        if not message:
            raise EOFError('socket closed')
        while not message.endswith(suffix):
            data = sock.recv(1024)
            if not data:
                raise IOError('received {!r} then socket closed'.format(message))
            message += data
        return message


def server():
    # start serving (listening for clients)
    port = 9001
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('localhost',port))

    # https://docs.python.org/3.8/library/threading.html#threading.Condition
    thread_cond = threading.Condition()

    while True:
        sock.listen(1)
        logging.info('Server is listening on port ' + str(port))

        # client has connected
        sc,sockname = sock.accept()
        logging.info('Accepted connection.')
        t = ClientThread(sockname, sc, thread_cond)
        t.start()

server()
```

Simple Client Example

PYTHON

```python
# client.py - a simple client

import argparse, socket, logging

# Comment out the line below to not print the INFO messages
logging.basicConfig(level=logging.INFO)

def recv_until(sock, suffix):
    """Receive bytes over socket `sock` until we receive the `suffix`."""
    message = sock.recv(1024)
    if not message:
        raise EOFError('socket closed')
    while not message.endswith(suffix):
        data = sock.recv(1024)
        if not data:
            raise IOError('received {!r} then socket closed'.format(message))
        message += data
    return message


def client(host,port):
    # connect
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host,port))
    sock.setblocking(True)
    logging.info('Connect to server: ' + host + ' on port: ' + str(port))

    msg = recv_until(sock, b"\n").decode('utf-8')
    logging.info('Message: ' + msg)

    sock.send(b"Hi server!\n")

    # quit
    sock.close()


if __name__ == '__main__':
    port = 9001

    parser = argparse.ArgumentParser(description='Client')
    parser.add_argument('host', help='IP address of the server.')
    args = parser.parse_args()

    client(args.host, port)
```

Development Environment

- Use the development environment setup for VS Code.
- You can test the server/client all locally (within the VM).
- Use the split feature to have three terminal windows open.

Submit

- ZIP'd file with the following:

- Python scripts, but no *pycache* or any IDE files please.
- Original RFC + Updated RFC.
- Testing output showing the protocol working as intended.

Version 1.0s21
Last updated 2021-02-19 13:31:13 -0800