CPRG 260 Final Project

1 Scenario

Fred is dead. Everyone is dead. During one of your scavenging missions as the sole survivor of the nuclear apocalypse, you've found a worn out server with some unknown program on it. The barely legible label reads "Tic Tac Toe" - a meaningless phrase in a strange language.

After some preliminary testing you've figured out this server is running a game of sorts but needs a client to function. Using your expert python skills you set out to make the client.

1.1 Server Details

After doing some analysis on the server binary, you've discovered that it is ran using the following syntax:

```
./tictactoeServer [PORT]
```

Where PORT is an optional command line argument specifying the port the server should bind to. If none is provided, it will default to 6969.

After being invoked, the server will wait and listen to incoming connections - it can only handle one client at a time. When a client connects, the transmission follows a client-server model. Using the implemented network protocol, the client sends requests and the server responds. The server handles keeping track of the game state and CPU opponent. Your client needs to handle its own main game loop and board rendering, requesting from the server where appropriate.

1.2 Client Requirements

There is a lot of freedom in how your client functions but at minimum the features required are as follows:

- On startup, the client will read a file in the same directory called tictac.ini containing the configuration options for your client. The syntax of this file is detailed below in section 1.4. If this file is missing or has incorrect syntax, the client should print an error message and exit.

- A main menu needs to implemented. When started, a banner and the options should be displayed. The options are: new game, load saved game, load configs, show score and exit.
- Functionality for loading and saving games. See section 1.3 for more details.
- The cumulative score of the CPU and human wins are recorded in a text file with time stamps.

The client is called as follows:

```
./python3 tictacClient.py
```

1.3 Network Protocol

The server keeps track of the game state and handles the CPU opponent. To facilitate the game, the client sends packets in the form:

```
HEAD:BODY
```

Where *HEAD* is the four character header of the packet and *BODY* is the body of the packet. Note that some packets do not have a body. For these packets, the colon can be left off.

Below is a list of all the packets used in the protocol:

- NEWG
  This packet has no body and causes the server to initialize a new game board and randomly determine if the CPU or player moves first (is X). The reply is a BORD packet.
- ENDG
  This packet has no body and causes the server to end the current game and reset the board. The reply is an OVER packet with the body as: no winner and the current state of the board.
- CLOS
  This packet has no body and causes the server to close the connection to the client. The reply is a CLOS packet.
- MOVE
  This packet is sent to the server during an active game to signify the player has moved. The body is the coordinates of the move in the form of *x,y*. Ex. MOVE:1,1 places the player's move in the middle of the grid. The server handles error checking and will return an EROR packet if there is an error. If there is no error, the server will return either an OVER packet if the game is over or a BORD packet if the game is still ongoing.

- BORD
  This packet is sent to the client when the board has been updated (after a move or load). The body is nine comma separated integers indicating the board state. 0 is X, 1 is O and 2 means the space is empty. For example, BORD:2,2,2,2,0,2,2,2,2 means that X is in the middle space and the rest are empty.

- LOAD
  This packet is sent to the server to load a saved game. The user should provide a file name with the saved game as its contents before this packet is sent. The body of this packet is the character the player is (X or O) and nine comma separated integers representing the board state as above. For example: LOAD:X,0,1,0,2,1,2,1,2,1 loads a game board where the client is X. The reply is a BORD packet with the loaded game state.

- EROR
  This packet is sent by the server to indicate something went wrong. The body is the error message. UNKNOWN CMD is sent when the packet header sent by the client is unknown. BAD MOVE is sent when the MOVE packet contains an invalid move. NO GAME is sent when the client is trying to MOVE before a game has started.

- OVER
  This packet is sent by the server to indicate the game is over. The body contains the winner (if there is any) and the current state of the board. C means the client won the game, S means the server did and N means no one won. For example: OVER:C,0,0,0,1,1,2,2,2,2 is sent when the client won playing as X.

1.4 Config File Syntax

The client loads a configuration file on startup or if the option is selected from the main menu. The config file looks like the following:

```
[BASIC]
saveDir=/tmp/savedGames
score=/tmp/scores.txt
[NETWORKING]
serverIP=127.0.0.1
port=6969
[FILTER_SCORE]
scoreTimeFrom=Nov 10 07:00
scoreTimeTo=Nov 16 07:00
```

Where *saveDir* is the directory containing the saved games. *score* is the file containing the cumulative score. *serverIP* is the IP address of the machine running tictactoeServer and *port* is the destination port of the client. The fields in FILTER_SCORE specify the time range that "Display score" counts. Your script needs to parse this file correctly and apply the options appropriately.

Hints:
- This is trickier than project 2; you don't have the code for the server and will need to handle your own game logic.
- Print out the raw packets you receive, this will help you decipher the network protocol.
- Don't be afraid to send arbitrary packets to see how the server behaves. You can also use any networking analysis tools you learned about in other classes.

1.5 Conditions

- If you made any code that we haven't covered in class, it NEEDS to be documented properly. Failure to do so will result in a deduction of grades. You can document in the comments of your python file.

2 Requirements

*ensure you have read course policies in D2L shell
Required python aspects for this project:

- Python script tictacClient.py.

3 Rubric

3.1 Python component

Your code will be graded per each criterion. Marks will be deducted if the implemented function deviates from the specification. All interpreter errors should be caught and handled appropriately!

Criterion	**Weight**
Main menu and game loop	10
Save/load game	10
Parse config file	10
General networking and score keeping	10
Error handling	10
Coding style and documentation	10
TOTAL	60

*Your instructors have seen so much code in their careers that at this point it is extremely easy to determine when someone is cheating. Do yourself a favour and don't cheat. 40% would be a tremendous hit to your final grade.

Python component	60
TOTAL	60