SWE 545 Distributed Systems Programming

Tic Tac Toe Project

Erhan Hasdağlı

2014719150

Motivation behind design decisions

This project has been implemented by using SimpleXMLRPCServer.According to my knowledge that I gained by researching on different web forums, “SimpleXMLRPCServer” doesn’t handle requests in parallel and also it isn’t session based. Briefly, if an object is served by “SimpleXMLRPCServer.register\_instance” method, it doesn’t create a new object on for each new client request. Because of those reasons, my main objective is to create a class which aims to manage threads that has game objects inside of them. In a nutshell, I created separate object for every client. Hence, I achieved the kind of session based management of which SimpleXMLRPCServer lacks. I used [this](https://gist.github.com/rpip/5608979) as a game engine although I changed most part of it.

XMLRPC API Documentation

1. **GameEngine.py**
   1. *Game Class*

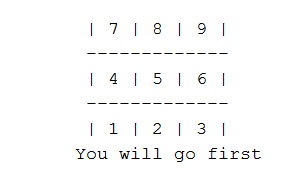
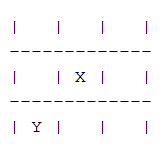
This class/module is taken from the link given above to provide base engine to the Game. It is internal to GameServer.py.

Any **client doesn’t have to know internals**. It is shown here to give insight game’s internal mechanism.

* + 1. Game Class Functions

**def print\_board:**

This function returns the current Tic Tac Toe board

**def help:**

Returns instructions about the game

**def is\_winner:**

Determines if there is a winner of the game according to current board status.

**def get\_bot\_move:**

Chooses best move place on the board for the bot and returns it

**def is\_space\_free:**

Returns if the requested move has already done

**def make\_move:**

Assigns the designated move to the board

**def choose\_random\_move:**

Chooses next move for the bot

**def start\_game:**

Starts the game and returns the instructions

**def check\_valid\_move:**

Checks if integer is given between 1-9 and space is free for intended move

**def make\_moves:**

Makes put moves on the board for bot and user and decides if one is winner. Returns boolean indicating is game still going on and message to display on the client side after move

1. **GameServer.py**
   1. *ThreadManager Class*

**This is the only class that is exposed to client** using server.register\_instance of the RequestHandler.

Main job of this class is create a thread for each client and manage them.

* + 1. *Thread Manager Class Properties*

**uniqueId :**

This variable is unique for each client.This is generated by using getrandbits and send back to client for future use.

**threadsDictionary:**

This dictionary holds the unique id as a key and thread as a value.

**dictionaryLock:**

This is used for preventing ambigious data in threadsDictionary.

* + 1. *Thread Manager Class Functions*

**def authenticate(self):**

Creates a unique id and returns to the client. Any client must call this function before starting game.

**def start\_new\_game(self,uniqueId):**

Starts the game and create a thread for each client game.

**end\_game(self,uniqueId):**

Finishes the game and remove client id from dictionary.Also nullify thread objects to make it ready for Garbage Collector.

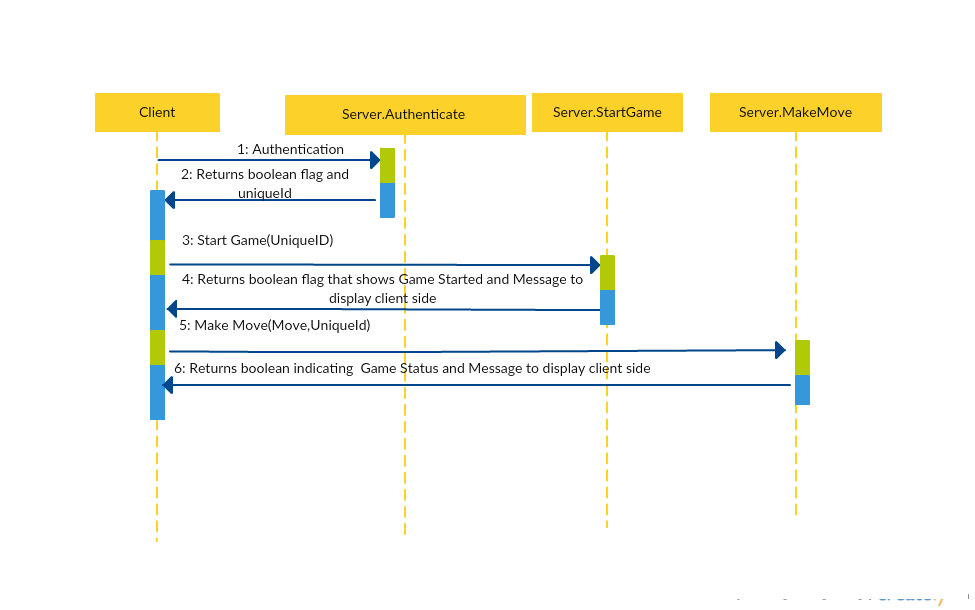
**def make\_move(self,move,uniqueId):**

Makes intended moves for client and bot. Client should provide its move and uniqueId as an input to this function.

2-2-1- WorkerThread Class

For each thread this class is created by ThreadManager class mentioned at section 2.Main responsibility is to call Game class mentioned at section 1.This class is not exposed to the client.

**3-Flow of Events**

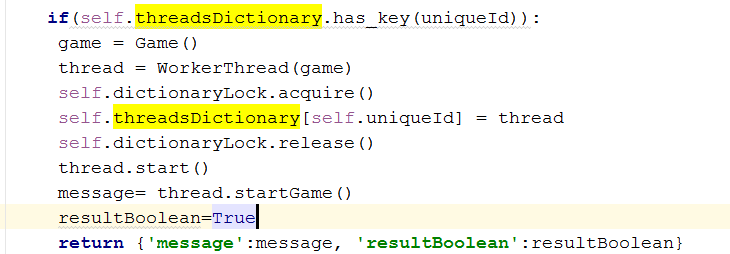


**4-Exceptions/Faults**

|  |  |  |
| --- | --- | --- |
| Code | Definition | Condition |
| 500 | Internal Server Error | It is a generic error message, given when no more specific message is suitable. |
| 404 | Destination Not Found | When server doesn’t run |
| 502 | Connection Timed Out | When server is under heavy traffic |
| 11 | Authentication Failed | When unauthenticated client tries to start game |
| 12 | Game Limit | When there are 50 players in the game. |
| 13 | Unexpected Input | When client input a non-numeric value or a value that is out of range |

**5-How multiple games are handled simultaneously?**

As mentioned above, ThreadManager class creates thread for each client. Each thread has its own Game object.Hence,they operate on different context. Below is an example code that starts the game.



**6-Couple of Scenarios**

* 1. **Success Case**

Client first calls Server. Authenticate and then starts game by calling Server.StartGame

* 1. **UniqueId Missing Case**

Client calls Server.StartGame without providing UniqueId or providing mis-matched unique id.Server raises uniqueId Fault/Exception

* 1. **Server Threshold Exceeded Case**

Client tries to authenticate when 50 clients are play the game.Server raises GameLimit exception.

* 1. **Wrong Input Case**

Client tries to call Server.MakeMove method with non-numerical input or input that is out of range. Server raises Unexpected Input exception/fault.