Homework 1

Hadachi&Lind

October 25, 2017

Must Read:

1. Deadline for doing homework is 3 weeks starting from now "2017.10.25"

   due date is: 2017.11.15 5:59:59 EET
2. For any delay in submitting the homework they be will a penalty (-5% of the grade for every day delay)
3. You can work in groups of maximum 4 people
4. Python 2.7 only
5. You should create a public git repository for your project and share it.
6. Be aware that the git should contain a time line progress of your implementation in case there is only one big commit at the end you will be penalized
7. You are allowed to use any fancy open source library that will help you in your design and development. **Except for network handling**. This means, you have to provide your own implementation of the application layer protocol based on transport layer of your choice (UDP, TCP, or Both)
8. Be aware that in case you use any ready product or source code of any existing application it will be considered as plagiarism.

1 Introduction

The main objective of this homework is to demonstrate how to handle transport layer protocols (TCP, UDP or Both) implementing you own application layer protocol. Therefore, we will implement a network based multi-player game.

1.1 Concurrent Sudoku Solver

Have you tried solving Sudokus before ? Did you Solve them by hand on the paper. Nowadays, the classic Sudoku game is ported to many platforms especially mobiles. As you can notice, sudoku is mostly single player game. In case you are unfamiliar to Sudoku solving we advise you to try it out. Therefore, in this homework we will try to create a sudoku, where many players can play on the same sudoku puzzle (Concurrent Sudoku).

2 Description

Our aim is to make the process of solving Sudoku concurrent, so that several players could contribute in solving the same Sudoku table at the same time. Just like in real world several persons would look at single Sudoku, and each person could suggest the next improvement (next guessed number and position). The order of the players in this case is not determined - each one can modify the Sudoku table at any time. Certainly this leads to several concurrency issues:

1. several players can suggest the same improvement

2. players can alternate the previously set guesses.

Your task is to program the application that could allow the concurrent Sudoku solving for several players (at least 2) over the network.
To create such application, it involves many challenges like handling concurrency, dealing with network connections, and building the GUI.

3 Win/Lose policy

In general, the classical win/lose policy in single-player Sudoku solving is simple: player wins once Sudoku is solved no matter how long it took him to do so. In case, there is time limit - player loses the moment Sudoku is not solved in the appropriate time. For **our homework**, we will adopt the following win/lose policy:

- for each correct guess (correctly suggested number and position) a player gets $+1$ score
- for each incorrect guess player gets -1 score.
- and no time constraint

Once Sudoku is solved the player with highest score wins, and the other players lose. The playing tempo and order are not set: players can suggest guesses at any moment as in race conditions (the fastest and smartest will win).
We won't complicate our server-side logic too much; therefore, we do not keep the player's state once the player leave. In this case, once the player join *"Sudoku solving session"* he/she should not leave, otherwise the collected points are lost (re-joining the left session player should start from 0 points). In case the players leave the session before Sudoku is solved - the last active player automatically wins.

4 General Network Architecture

The general architecture is illustrated in figure 1 and it is structured as follows:
Dedicated Game server is responsible for:

- creating game sessions on demand and generating Sudoku to be solved
- handling game sessions and corresponding player connections.
- counting players activities giving the scores for correct guesses or reducing scores for bad guesses.

Game client is responsible for:

- handling user input
- showing dialog: provide player nickname
- showing dialog: provide server address
- showing dialog: chose to join existing session or to start the new one
- rendering current game state (Sudoku)
- rendering score of the players
- communication to dedicated server

5 Requirements

The following use cases should be covered by your applications.

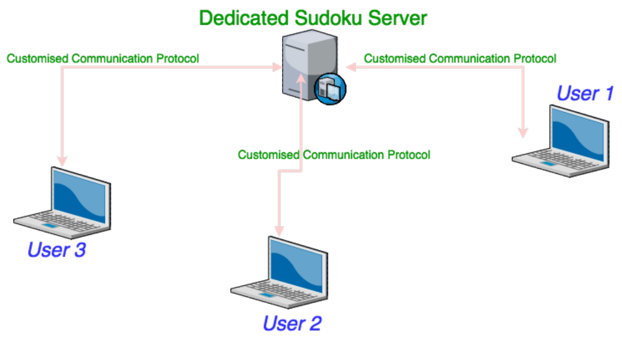
Dedicated Sudoku Server
Customised Communication Protocol
Customised Communication Protocol
User 1
Customised Communication Protocol
User 3
User 2

Figure 1: Overview of the general architecture. Here "Customized Communication protocol is your-suggested application layer protocol on top of TCP or UDP

5.1 Starting dedicated server

Before starting the server user has to specify what port and address the server has start listening on. Once started the server cannot alternate the listener socket.

5.2 Starting client application

1. User opens an application.
2. Application shows a "Enter Nickname" dialog.
3. User may specify new nickname or select one of the previously specified nicknames.
4. Application checks if the specified nickname is not longer than 8 alphanumeric characters (empty strings not allowed, spaces not allowed).
5. Once nickname is specified and is valid, user may proceed to next dialog Enter Sudoku server address" dialog. Otherwise nickname must be fixed to pass validation.

5.3 Connecting to dedicated server

1. Application shows "Enter Sudoku server address" dialog
2. User has to provide the address of the socket the dedicated server is listening on
3. Once the server address is specified user may connect to server.
4. In case of successful connection user may proceed to next dialog "Multiplayer Game"
5. In case of connection errors, an application must show a human-readable error message: "Connection refused" or "Server not Found" etc.

5.4 Multiplayer Game Dialog

1. Once successfully connected to Sudoku server
2. Application must fetch the list of exiting sessions and show them to user.
3. User may select any of the fetched sessions to join one. Or user may click create a new session.
4. In case user decides to join existing session, an application proceeds to "Joining Sudoku Solving Session" scenario
5. In case user decides to create new Sudoku solving session, an application proceeds to "Creating new Sudoku solving session" scenario

5.5 Creating new Sudoku Solving Session

1. Application shows "Creating new Sudoku Solving Session" dialog.
2. User provides the number of desired players and clicks create
3. Application sends request to server
4. Server registers new session and associates an instance of Sudoku to be solved within this session
5. Once session is registered, server replies to client with session id
6. Client application proceeds to Gameplay scenario

5.6 Joining existing Sudoku Solving Session

- User selects the desired session from the list of available sessions on the server.
- Application sends the request to server asking to join the selected session
- Server replies with accepting or not accepting the player for the selected session.
- In case server allows a player to join the session and application proceeds to Gameplay scenario.
- In case server does not permit player to join (for example session is full) an application returns to Multiplayer Game Dialog so the user can join another session or create a new one.

5.7 Leaving Sudoku Solving Session

The moment of leaving the Sudoku solving session, we can distinguish two cases:

- The Sudoku puzzle was solved and we terminate the game and announce the winner to all the players.
- One of the players has left the game, we close the player's session and if only one player left playing; then, we declare him/her the win.
- Irregardless of the game result - server must take care of empty session (those left by all player - 0 players connected). Server must remove those sessions automatically.

5.8 Gameplay

5.8.1 Gameplay:Waiting for players

Once player joined, an application should still wait till the other players joined too. The gameplay should only start once all the players are joined.

5.8.2 Gameplay:Rendering

- Once player joined the session the application must: a.) fetch the current state of the Sudoku table b.) fetch the current scores for all the players in this session.
- Redraw both Sudoku table and Scores table
- Application keeps open the communication channel with the Sudoku server till client decides to close the application or leave the session.
- Server notifies the client application about the Sudoku changes.
- Application must listen for change notifications from the server side.
- Once notification is received and application must a.) fetch the current state of the Sudoku table b.) fetch the current scores for all the players in this session.

- Redraw both Sudoku table and Scores table (score table must contain all player names in this session and their collected scores).
- Once the winner is known, server must inform all the active players about the winner in the next notification.
- Application must render message "You win!" for winner or "Player X won!" for others.
- Application must leave the active game session and proceed to "Multiplayer Game Dialog" to let user join another session or create another new one.

5.8.3 Gameplay:User input

- User input is active all the time while Sudoku table is shown
- User can pickup the Sudoku table cell and suggest number
- Be aware some of the Sudoku cells are immutable with predefined numbers, an application should not allow modify those cells.
- Application does not alternate the clients local Sudoku table (the one showed to user) instead the suggested change (guess) is sent to server for processing.
- Server receives the guesses from different players in natural time order (fastest player first).
- Server resolves the collisions in case multiple players suggest the same change - the earliest one gets processed and the rest ignored.
- Irregardless of who was authoring the change the state of the Sudoku table is updated and the notification change is sent to all connected clients.
- In order to give score for each accepted change (guess) server checks if the change suits the correct solution for the particular Sudoku. In case of a correct guess the server gives $+1$ score to the player who introduced the change. In case the player did introduce the wrong guess the server count -1 score to corresponding player. After each modification of scores table server notifies all the connected players.

6 Implementation

You are free to use any third party libraries for GUI, user input processing etc (except of network handling - you are only allowed to use the socket api of Python) when implementing the application (as long as the communication protocols are respected). You can choose any GUI design as long as the game requirements are respected. The use of threading is up to you and you can avoided them as long as other requirements are preserved.

7 Deliverables

They will be two major deliverable for this Homework:

First: You have to submit your Team members names including you and also the public git repository link via the Moodle, latest 3st November at 12h00.
Moodle Link

Second: A report that includes the following:

- User manual explaining setup process: Client, Server, and (other possible dependencies required).
- User manual explaining the user interface and how to play the game.

Third: The source code of your implementation.
Concerning the implementation, we will consider only the source code from your public git repo provided. Note, any modification done after the due date to the git won't be considered. Concerning the report, it should be PDF document and submitted in the course website by one member of the team.
Submit Link

8 Grading

Total points for this Homework is 20 pts, and its breakdown is as follows:
• 6 pts: The user manual document
• 14 pts: The implementation:[– 2pts: code is functional (and does run) – 6pts: the communication protocols are used correctly – 4pts: requirements were respected – 2pts: code is documented]