Programación de Sistemas de Telecomunicación
Práctica 2: Universidades y Mazmorras en red

GSyC

Noviembre de 2020

1. Introducción

En esta práctica deberás implementar una aplicación cliente-servidor a través de la que se pueda jugar en red al juego implementado en la primera Práctica.

La aplicación estará dividida en dos partes principales, server.py dónde se gestionarán las partidas que se vayan jugando y los jugadores que se vayan uniendo a dichas partidas y; client.py dónde se implementará la funcionalidad relativa al cliente (Unirse al servidor, pedir crear o unirse a una partida existente, salir del servidor... y la recepción de los mensajes por parte del servidor.)

2. Especificación del cliente

2.1. Argumentos

Al ejecutar el programa se podrán pasar opcionalmente los siguientes argumentos

`-p` Este argumento irá acompañado de un número entero (Por ejemplo -p 3) y permitirá establecer el número de jugadores de la partida. En el ejemplo propuesto, habrá un total de 3 jugadores en la partida. El programa también deberá permitir recibir como argumento `--players` (–players=3). En ningún caso el número de jugadores podrá ser inferior a 1 ni superior a 4. Tampoco se permiten valores no enteros para dar valor a este argumento.

`-s` Este argumento irá acompañado de un número entero (Por ejemplo -s 5) y permitirá establecer el número de niveles del juego. En el ejemplo propuesto,los jugadores tendrán que superar un total de 5 niveles. El programa también deberá permitir recibir como argumento `--stages` (–stages=5). En ningún caso el número de niveles puede ser inferior a 1 o superior a 10. Tampoco se permiten valores no enteros para dar valor a este argumento.

El programa por lo tanto debe poder iniciarse de esta forma.

- `python main.py -p 3 -s 5`
  Deberá iniciar una nueva partida para 3 jugadores con 5 niveles.
- `python main.py --players=3 -s 5`
  Deberá iniciar una nueva partida para 3 jugadores con 5 niveles.
- `python main.py --players=3 --stages=5`
  Deberá iniciar una partida para 3 jugadores con 5 niveles.
- `python main.py`
  Deberá iniciar una nueva partida para un jugador con un nivel.
- `python main.py -p 5 -s 3`
  Debe avisar de que el número de jugadores es incorrecto y finalizar la partida.
- `python main.py -p 5 -s 11`
  Debe avisar de que el número de jugadores y el número de niveles son incorrectos y finalizar la partida.

Si el argumento no recibiera un número o el número recibido fuera incorrecto según las especificaciones anteriones, el programa lanzara una excepción y terminará.

Adicionalmente, es obligatorio que el programa reciba los siguientes argumentos:

-n Este argumento irá acompañado de un nombre o nick, que representa el nombre del jugador. El programa también deberá permitir recibir como argumento `--name` (–name=david). El nick puede tener cualquier longitud y puede contener números.

-i Este argumento irá acompañado de la ip dónde se encuentra el servidor. El programa también deberá permitir recibir como argumento `--ip` (–ip=127.0.0.3). Si no se proporciona el argumento, el valor por defecto será 127.0.0.1.

-o Este argumento irá acompañado del puerto dónde se encuentra el servidor. El programa también deberá permitir recibir como argumento `--port` (–port=6123). Si no se proporciona este argumento, el valor por defecto será 8080.

De estos tres últimos argumentos, sólo es necesario comprobar que el puerto sea un número entero. Si no se proporciona un nombre para el jugador, el programa deberá finalizar.

2.2. Ejecución

A continuación se muestran algunos ejemplos de la primera conexión que realizaría un cliente a un servidor que se está ejecutando en local (127.0.0.1) en el puerto 8080.

```
$ ./python client.py --stages=5 --players=2 --ip=127.0.0.2 --port=8080  --name=David
        Could not connect to the server. Are you sure you have provided the correct ip and port

--------------------------- EJECUCIÓN DEL PRIMER JUGADOR ---------------------------
$ ./python client.py --stages=5 --players=2 --ip=127.0.0.1 --port=6123 --name=david #First play
        Welcome to the server. Choose one of this options:
                  1.- Create game
                  2.- Join game
                  3.- Exit
        Your option: 1
        #if 1 is selected:
        ******************************************************
        Choose a character
        ******************************************************
        1.-The bookworn -> Stats: 50HP and 2DMG and 12LD
        2.-The worker -> Stats: 5HP and 2DMG and 10LD
        3.-The whatsapper -> Stats: 20HP and 10DMG and 5LD
        4.-The procrastinator-> Stats: 30HP and 10DMG and 3LD
        ******************************************************
        Choose one option: 1
           Waiting for other players to join the game #In this example, we do not start the gam
        #The second player joins the game
                     ************************
             *       STAGE 1        *
             ************************
            ---- CURRENT MONSTERS ----
      +++++++++++++++++++++++++++++++++++
     Theoretical class:  Stats: 8HP and 3DMG
     Theoretical class:  Stats: 8HP and 3DMG
     Theoretical class:  Stats: 8HP and 3DMG
      +++++++++++++++++++++++++++++++++++
        Bookworn (david). What are you going to do?: a
        The Bookworn (david) did 2 damage to Theoretical Class.Theoretical Class has 6 hp left.

------------------- EJECUCIÓN DEL SEGUNDO JUGADOR -----------------------------
$ ./python client.py --stages=5 --players=2 --ip=127.0.0.1 --port=6123 --name=tamara #Second pl
        Welcome to the server. Choose one of this options:
                  1.- Create game
                  2.- Join game
                  3.- Exit
```

```
        Your option: 2
        #if 2 is selected:
        -----------------------------------------------
        Available games
        -----------------------------------------------
        1.- Players: 1/2
        -----------------------------------------------
        Choose one game to join: 1
        ***************************************************
        Choose a character
        ***************************************************
        1.-The bookworn -> Stats: 50HP and 2DMG and 12LD
        2.-The worker -> Stats: 5HP and 2DMG and 10LD
        3.-The whatsapper -> Stats: 20HP and 10DMG and 5LD
        4.-The procrastinator-> Stats: 30HP and 10DMG and 3LD
        ***************************************************
        Choose one option: 1
             ************************
             *       STAGE 1        *
             ************************
            ---- CURRENT MONSTERS ----
      ++++++++++++++++++++++++++++++++++++
     Theoretical class:  Stats: 8HP and 3DMG
     Theoretical class:  Stats: 8HP and 3DMG
     Theoretical class:  Stats: 8HP and 3DMG
      ++++++++++++++++++++++++++++++++++++
The Bookworn (david) did 2 damage to Theoretical Class.Theoretical Class has 6 hp left.
Bookworn (tamara). What are you going to do?:
```

Consideraciones a tener en cuenta cuando un jugador se une a una partida:

- Hasta que un jugador no elige personaje no se considera que se ha unido a la partida. Por lo tanto, puede que en el proceso de selección otro jugador se conecte y le quite el hueco al primero. Hay que controlar dicha situación. Por ejemplo, una vez se elija el personaje se puede comprobar si la partida está completa.
- Hay dos formas de listar las partidas disponible. La primera es listarlas todas (incluso las completas) y no permitir que el cliente se una a ellas o listar únicamente aquellas partidas que no están completas.
- Si proporcionamos una ip y un puerto donde no se está ejecutando ningún servidor se lanzará la excepción (de forma automática por Python) ConnectionRefusedError. Es necesario controlar dicha excepción para que el programa finalice grácilmente.
- Se recomienda la siguiente forma de almacenar la información de los clientes y las partidas en el servidor (Aunque en ningún caso es obligatorio seguir este modelo). Por un lado una variable numérica para controlar el número de partidas que hay (o ha habido en el servidor). Otra variable que será un diccionario en el que las claves serán el valor de la variable numérica de la frase anterior y el valor será la instancia de una partida. Por último, una variable que será un diccionario cuyas claves sean una tupla (ip, puerto) de cada cliente (lo que conocemos como address) y cuyos valores serán las id del "Game" en el que se encuentra dicho cliente.

```
id = 1
games = {1 : Game(), 2: Game()}
client_game = {(127.0.0.1, 6123): 1, (127.0.0.1, 6124): 1, (127.0.0.1, 6125): 2}
```

3. Servidor

3.1. Argumentos

`-p` Este argumento irá acompañado del puerto dónde se encuentra el servidor. El programa también deberá permitir recibir como argumento `--port` (–port=6123). Hay que comprobar que el número recibido como argumento es un número entero. Si no se recibe este argumento, el valor por defecto del puerto será 8080.

3.2. Ejecución del servidor

```
$ ./python client.py --port=6123
Server started at 127.0.0.1:6123
(WELCOME) david joined the server
(CREATE) david created a game
(WELCOME) tamar joined the server
(JOIN) tamara joined david's game
(START) david, tamara started a game
#No es necesario imprimir en el servidor los mensajes de la partida. Esos mensajes solo se envi
(GAMEEND) david, tamara game ended. They won.
O
(GAMEEND) david, tamara game ended. They lost.
# Si alguno de los jugadores se desconecta durante el proceso
(EXIT) david disconnected.
# Cuando un jugador se desconecta se debe desconectar al resto de los jugadores
(DC) tamara was disconnected.
```

3.3. Protocolo de comunicación

Los mensajes serán enviados entre el cliente y el servidor a través de diccionarios que contendrán el tipo de mensaje y la información necesaria para el correcto funcionamiento de la aplicación. A continuación se detallan los mensajes básicos que deben utilizarse, defiendo las claves de los diccionarios y los valores que se almacenan en dichas claves.

Mensaje `Join`

Mensaje que le envía el cliente al servidor al realizar la conexión.

`Protocol`	`Name`

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Nick`: El nombre del cliente

Mensaje `Welcome`

Mensaje que le envía el servidor al cliente una vez se conecta.

`Protocol`	`Message`	`Options_Range`

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Message`: El menu para crear o unirse a una partida o abandonar.
- `Options_Range`: Una lista con el rango de las opciones disponibles ([1, 3])

Mensaje `Send_Server_Option`

Mensaje que le envía el cliente al servidor con una opción del menú anterior.

Protocol	Option	Players	Stages

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Option`: La opción elegida.
- `Players`: Número de jugadores.
- `Stages`: Número de niveles.

Mensaje `Choose_Character`

Mensaje que le envía el servidor al cliente cuando crea una partida o se une a una partida para seleccionar un personaje.

Protocol	Message	Options_Range

en donde:

- `Protocol`: El tipo de mensaje.
- `Message`: El menu con los diferentes personajes a seleccionar.
- `Options_Range`: Una lista con el rango de las opciones disponibles ([1, 4])

Mensaje `Send_Character`

Mensaje que le envía el cliente al servidor cuando selecciona un personaje.

Protocol	Option

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Option`: La opción del personaje elegido.

Mensaje `Server_Msg`

Mensaje que le envía el servidor al cliente con texto.

Protocol	Message

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Message`: El texto.

Mensaje `Your_Turn`

Mensaje que le envía el servidor al cliente para decirle que es su turno y pedirle una acción.

Protocol	Message Range_Options

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Message`: El texto. El texto contendría algo similar a lo siguiente. Personaje (Nombre_Jugador) . What do you want to do? [a, s]:
- `Range_Options` Los comandos disponibles. [a, s]

Mensaje `Send_Character_Command`

Mensaje que el cliente al servidor con el comando seleccionado.

Protocol	Command

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Command`: El comando (a o s).

Mensaje `Send_Games`

Mensaje que le envía el servidor al cliente con la lista de partidas disponibles

Protocol	Message	Options_Range

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Message`: La lista de partidas
- `Options_Range`: Una lista con el rango de las opciones disponibles ([1, 2, 3, 4])

Mensaje `Send_Game_Choice`

Mensaje que le envía el cliente al servidor con la partida seleccionada.

Protocol	Option

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Option`: La partida seleccionada.

Mensaje `Send_Valid_Game`

Mensaje que envía el servidor al cliente si se ha podido unir a la partida seleccionada.

Protocol	Joined

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Joined`: True or False.

Mensaje `Send_End_Game`

Mensaje que envía el servidor al cliente cuando una partida finalizar (Tanto si es por derrota de los jugadores como por victoria).

Protocol	Win

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Win`: True or False.

Mensaje `Send_DC_Me`

Mensaje que le envía el cliente al servidor para informarle de que se desconecta. El servidor debe desconectar a todos los jugadores que están en esa partida.

Protocol

en donde:

- `Protocol`: Ell tipo de mensaje.

Mensaje `Send_DC_SERVER`

Mensaje que envía el servidor al cliente para desconectarlo.

Protocol	Reason

en donde:

- `Protocol`: Ell tipo de mensaje.
- `Reason`: Mensaje con la razón de la desconexión. Por ejemplo: david disconnected. The game can not continue.

3.4. Recomendaciones

1. Es recomendable añadir al juego un atributo donde se guarde el socket, el nombre de cada jugador y la dirección ip (address). De esa manera, cada vez que haya que enviar un mensaje a un conjunto de jugadores (la información del nivel, el movimiento de otro jugador, ...) podrá acceder a la propia partida para obtener dicha información. Ésto en combinación con lo especificado en el punto 2.2 sobre como guardar las partidas, le otorgaría todo lo necesario para el correcto funcionamiento del programa.
2. En algunas ocasiones, puede que necesite enviar dos mensajes de manera simultánea. Si fuera el caso, recuerde utilizar las funciones `send_one_message` y `recv_one_message` vistas en clase.
3. Comprueba cada mensaje que implemente. Resolver problemas a posteriori es más complicado que resolverlos cuando se está implementando un mensaje en concreto.
4. Utiliza el código desarrollado en clase de teoría como base para tu aplicación.

4. Condiciones obligatorias de funcionamiento

1. El programa finalizará cuando:
    1.1. Los argumentos sean incorrectos. (Cliente / Servidor)
    1.2. Se pulse Ctrl + C. (Cliente / Servidor)
    1.3. Los jugadores ganen. (Cliente)
    1.4. Todos los jugadores sean derrotados. (Cliente)
    1.5 Se pulse la tecla Enter (Servidor).
    1.6 No se pueda realizar la conexión. (Cliente)
2. Los programas deberán escribirse teniendo en cuenta las consideraciones sobre legibilidad y reutilización del código que hemos comentado en clase.
3. Puede utilizar variables globales para compartir información entre los diferentes procesos tal y como se ha visto en clase de teoría. El uso de variables globales para cualquier otra cosa será penalizado.
4. Los programas deberán ser robustos, comportándose de manera adecuada cuando no se arranquen con los parámetros adecuados en línea de comandos.
5. Es **obligatorio** controlar todas las excepciones que puedan surgir durante el transcurso del programa: argumentos inválidos, selección de opciones incorrectas, ctrl + c cuando el programa se está ejecutando, etc.
6. Debe seguirse el modelo de salida mostrado en las secciones 2 y 3 de este enunciado.

5. Pautas de Implementación

- Es muy importante que aquella funcionalidad estrechamente relacionada con el juego en si se implemente dentro de server.py o de game.py. En ningún momento el cliente tendrá acceso a game.py ni podrá conocer la situación específica de su partida si no se la envía el servidor.
- Crea funciones o métodos para gestionar la recepción y el envío de mensajes, tal y como se ha visto en clase.
- Se pueden crear excepciones personalizadas según se estime oportuno. Por ejemplo, una excepción para lanzar un error de argumentos es recomendable.
- Se pueden utilizar los módulos que se consideren oportunos.
- Recuerda evitar duplicar código. Si hay más de 5 líneas que se repiten a lo largo de tu programa, es recomendable que crees una función.
- Recuerda que cada vez que utilizas la función `input()` se está leyendo un string. Cada vez que se lee un argumento se está leyendo también un string.

6. Entrega

La práctica puede realizarse en grupos de hasta 3 personas. Sólo hará falta entregar la práctica una vez. Deben aparecer los nombres y apellidos de todos los autores en el fichero client.py en forma de comentario al principio del fichero. Se podrán subir los ficheros de uno en uno, por lo que no es necesario comprimirlos.

El límite para la entrega de esta práctica es el **Domingo, 20 de Diciembre a las 23:59** a través de la tarea correspondiente en el AulaVirtual.

La **Prueba de Laboratorio 1** correspondiente a esta práctica se realizará en el aula de prácticas habitual el **Jueves 17 de Diciembre a las 11:00**