Redes de Computadoras

Curso 2021

Grupo 6

Informe - Obligatorio 2

Autores:
Guillermo Toyos
Federico Vallcorba
Santiago Olmedo

Supervisor:
Eduardo Grampín

October 10, 2021

Contents

1 Problema a resolver

Para el obligatorio se propuso desarrollar un juego en línea siguiendo la arquitectura cliente-servidor. El juego se basa en un mundo compartido, donde los jugadores coexisten. El mundo consiste en un mapa de $100m^2$ donde se mueven los jugadores, que tienen un rango visual de $15m$ (i.e los jugadores pueden ver a aquellos jugadores que estén a menos de $15m$ de distancia de ellos). Los jugadores se mueven sin detenerse a una velocidad de $1m/s$ en una de las cuatro direcciones cardinales (N, S, E y O). Los clientes controlan su jugador usando las teclas del teclado (W/A/S/D). Para evitar que los clientes hagan trampas, los clientes reciben solamente la información del mundo que se encuentra dentro de su rango visual.

Para resolver el problema se debió implementar una aplicación distribuida. El servidor debe contener la simulación del mundo y aceptar las conexiones de los clientes. Cada vez que un cliente se conecta, se debe instanciar un nuevo jugador, el cual se debe destruir una vez que el cliente se desconecta. Los clientes podrán indicarle al servidor si quieren moverse a una nueva dirección y el servidor deberá enviarle constantemente información actualizada del mundo que le rodea. Para la comunicación entre las dos partes fueron definidos dos protocolos: *CONTROL* y *MUNDO*.

1.1 Protocolo CONTROL

El protocolo *CONTROL* es utilizado por los clientes para crear la sesión de juego y controlar a su jugador. Utiliza el protocolo de transporte TCP conectándose al puerto *2021* del servidor. Cada comando finaliza con el carácter "\n", y los mismos siempre son iniciados por el cliente, algunos disparando una respuesta de parte del servidor y otros no. La primer parte del protocolo consiste en el login, el cliente declara su nombre que lo identificará en el mundo a lo cual el servidor debe responder **OK** o **FAIL**. Si se recibe **OK**, el cliente procede a indicarle al servidor el número de puerto que va a utilizar para recibir los mensajes del protocolo *MUNDO*. Si el servidor le responde **OK**, el inició de sesión fue exitoso. Tras esto, la conexión TCP quedará abierta indefinidamente y el cliente podrá enviar los comandos para indicar la dirección en la cual quiere moverse en el mundo.

1.2 Protocolo MUNDO

El protocolo *MUNDO* se utiliza para enviar el estado del mundo a cada cliente desde el servidor utilizando el protocolo UDP. Todos los mensajes empiezan con "WORLD timestamp \n PLAYER x y dir \n", donde timestamp es una marca que indica el momento en que se envío el mensaje desde el servidor. La siguiente linea indica la posición y dirección del jugador en el mundo. En los mensajes del protocolo MUNDO solamente deben aparecer la información (coordenadas y dirección) de los demás jugadores que se encuentran a menos de 15 metros del jugador. Esta información podrá estar incluida después de las dos lineas WORLD y PLAYER y seguirá el formato "nickname x y dir \n". Donde nickname es el nombre que identifica al jugador.

La siguiente imagen ilustra el intercambio de mensajes entre un cliente y el servidor para los dos protocolos:

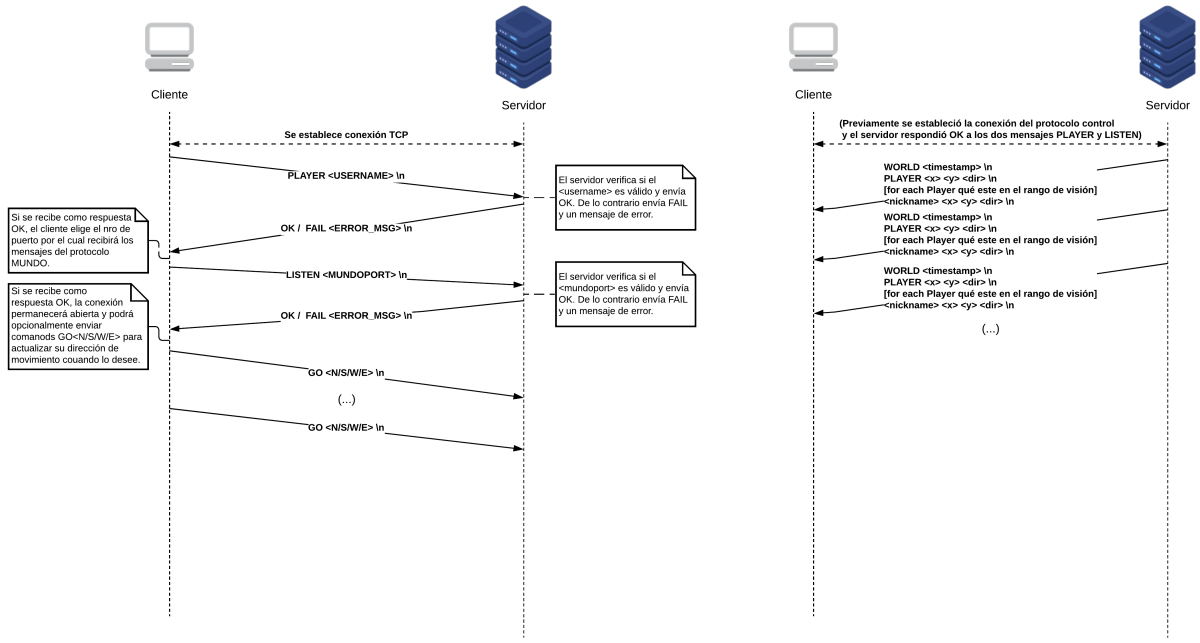

Figure 1: Intercambio de mensajes para el protocolo CONTROL (izquierda) y MUNDO (derecha)

2 Solución propuesta

Para resolver el problema se decidió implementar en Python[1] (con su librería de sockets [2]) tanto el cliente como el servidor. Para la comunicación entre las partes se implementan los protocolos de comunicación *CONTROL* y *MUNDO*. Es decir, habrá un servidor esperando que atenderá las conexiones de los cliente por el puerto TCP 2021 siguiendo el protocolo control y enviara a los clientes quienes iniciaron sesión exitosamente información del mundo con el protocolo *MUNDO*.

2.1 Servidor

Como todo clásico programa, este inicia su ejecución en la función *main()*. La cual crea dos hilos de ejeución: Uno que va a escuchar las conexiones entrantes por el puerto TCP 2021, utilizando la función *controlListener* y otro que se va a encargar de simular al mundo utilizando la función *mundoSimulator*.

La función *controlistener* crea un "socket servidor" de acogida en el puerto 2021 para esperar conexiones TCP por parte de los clientes. Al conectarse un cliente se crea un socket TCP de conexión y se crea un hilo que ejecuta la función *atenderCliente* con dicho socket pasado como argumento.

La función *atenderCliente* recibe mensajes del cliente, los procesa y le responde al cliente. Al principio recibe la información de login del cliente con su nombre, y se fija si es válido. Es decir, el formato es correcto, no hay ningún otro jugador ya instanciado con dicho nombre, no se supera el número máximo de conexiones por host y el nombre escogido no supera una cantidad de caracteres definida. Estos dos últimos controles se realizan para evitar abusos de parte del cliente. Si se pasan todos los controles se envía el mensaje OK al cliente, de lo contrario se envía FAIL y un mensaje de error. Luego, el servidor recibe el número de puerto que el cliente decide utilizar para recibir la información del MUNDO. En este caso se chequea si el puerto UDP ya está en uso para dicho host, y se le responde al usuario de acuerdo a esto. Esto se hace para que el cliente no reciba mensajes UDP para dos jugadores distintos en el mismo puerto.

Si el inicio de sesión es exitoso, se crea al usuario y se lo agrega al mundo. Luego, se crea un thread que ejecuta la función *mundoBroadcaster* con la información del usuario como argumento (username, hostname y puerto escogido para *MUNDO*) la cual implementará el protocolo *MUNDO*. Finalmente, el hilo que

ejecuta *atenderCliente* se queda esperando mensajes TCP con los comandos que envía el usuario para actualizar su dirección en el mundo con la función *procesarComandos*. Para hacer esto, a medida que llegan bytes al socket de conexión, *atenderCliente* va identificando los comandos que llegan y los coloca en una cola, luego ejecuta la función *procesarComandos* que procesa dichos comandos encolados y efectivamente actualiza el mundo.

Los hilos que ejecutan *mundoBroadcaster* son los responsables de enviar los mensajes con la información del mundo para los clientes siguiendo el protocolo MUNDO. Los parámetros que recibe dicha función identifican al socket del cliente (con el puerto que esté eligió) y su identificación en el mundo. Esta función envía la información del mundo y se bloquea por 0.01 segundos. Para enviar el mensaje, la función lee la información actual del mundo. Luego, por cada persona en el mundo con distinto username que el del usuario se fija si está en el rango visual mencionado en la sección anterior. Para el envío del mensaje se sigue el protocolo *MUNDO*. Dado que la cantidad de jugadores en el rango visual no está acotada, la función se asegura de dividir los jugadores en varios mensajes para que estos sean menores al tamaño máximo de un segmento UDP. Todos los estos mensajes tienen las primeras dos líneas obligatorias del protocolo y comparten el mismo timestamp.

Por último, el thread que ejecuta *mundoSimulator* es el responsable de desplazar los jugadores presentes en el mundo según su dirección y la velocidad del simulador: $1m/s$. La función se bloquea cada $0.01s$ cada vez que actualiza el mundo.

La implementación del mundo en el servidor es una estructura de datos de tipo diccionario donde las claves son los nombres de los jugadores en el sistema y las entradas son listas que contienen el hostname, puerto *MUNDO*,coordenadas y dirección de cada jugador. Dada que esta estructura es accedida concurrentemente, se implementó un mutex que asegura que un solo thread pueda modificar el mundo a la vez.

El siguiente diagrama ilustra el comportamiento del servidor. Cada rectángulo representa la actividad que se realiza (ejecutada por un hilo) y entre paréntesis la función que lo implementa. Las líneas gruesas representan un fork para realizar dicha actividad. Los rombos pequeños señalan que se deben cumplir señales y condiciones para avanzar. El circulo con una cruz indica el fin del hilo en eje-

cución y los dos triángulos un sleep de DeltaTiempo (0.01 segundos). El estilo del diagrama esta inspirado en los diagramas de actividad UML presentados por Fowler [3].

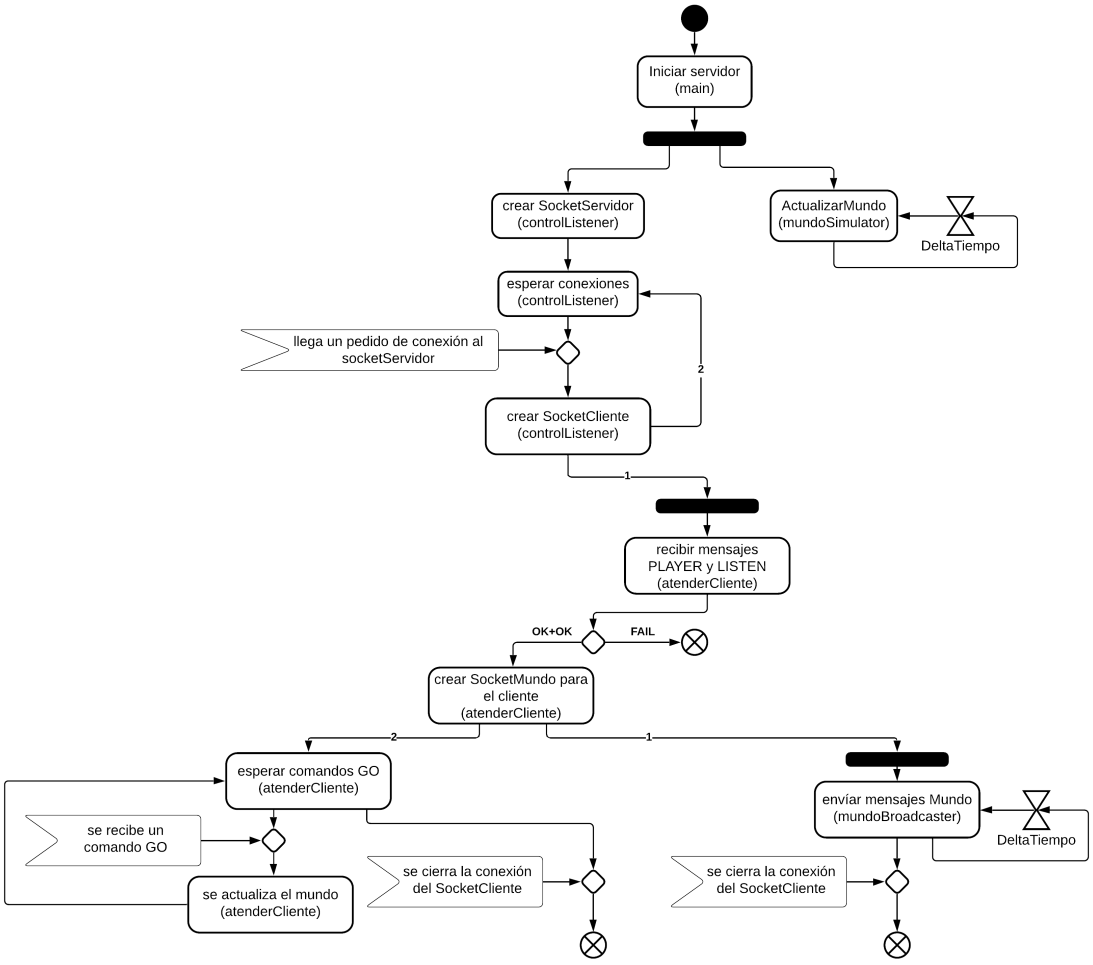

Figure 2: Diagrama de actividad del servidor

2.2 Cliente

Del lado del cliente, el programa puede dividirse en dos grandes estapas: El establecimiento de la conexión y login a traves del protocolo CONTROL y el recibimiento de mensajes *MUNDO*, su despliegue en pantalla, y el envío de comandos de dirección por *CONTROL*.

El programa comienza esperando que el usuario indique por input el nombre con que desea identificarse. Luego, se pasa al establecimiento de la conexión con el servidor con la función controlCliente. En primer lugar, la función crea un socket TCP con el que se conecta a la IP y puerto del servidor. Luego, se le envía el mensaje PLAYER y el nombre indicado. Si se recibe OK, se crea un socket UDP (con un puerto asignado por el sistema operativo) que es el que va a recibir la información del mundo y se le indica el número al servidor en el mensaje LISTEN. Si se recibe FAIL para cualquiera de los dos mensajes el programa termina. De lo contrario, el login fue exitoso y el cliente ya debería estar recibiendo mensajes de su posición en el mundo!

En la segunda etapa, se crea un listener (utilizando la librería pynput [5] que ejecuta la función KeyCommand cada vez que se lee una tecla del teclado. Si se lee una tecla de dirección (W/A/S/D) este indica por el socket de CONTROL la nueva dirección deseada al servidor. Por otro lado, si se lee la tecla Q se procede a cerrar el cliente. Luego se crea un hilo que ejecuta la función readWorld, este es el encargado de leer los mensajes enviados por el socket de MUNDO. Finalmente se pasa a ejecutar la función UpdateScreen la cual utiliza la librería turtle [4] para imprimir en pantalla la información del mundo de una manera más lúdica que un stream de texto en la terminal.

La función readWorld administra la estructura de datos que mantiene la información de los jugadores visibles y la posición del propio cliente. Updatescreen leerá de esta estructura para desplegar los jugadores en la pantalla. En esta dinámica de productor/consumidor, se implementó un semáforo que asegura la coherencia de la lectura. De lo contrario podrían desplegarse posiciones de jugadores de tiempos distintos al mismo tiempo.

Dado que el protocolo control utiliza UDP, la función readWorld se asegura de que los mensajes recibidos no lleguen fuera de orden. Para ello, se mantiene el timestamp más reciente que se ha recibido y los proximos mensajes que lleguen con un timestamp mayor, los descarta, si tiene el mismo timestamp hace un "append" a la estructura de datos con la información de los jugadores visibles recibida (y la del cliente en sí) y si es mayor actualiza el timestamp y reemplaza la estructura de datos con el nuevo contenido recibido.

La función updateScreen ajustan ciertos parámetros de la librería Turtle [6] y se diseño algoritmo relativamente eficiente para conseguir que el movimiento de los

jugadores se muestre de forma fluida. A grandes rasgos, el algoritmo consiste en mantener una estructura con los jugadores que se están mostrando en pantalla y en cada iteración se lo compara con la estructura de datos que modifica readWorld, los jugadores nuevos se instancian en las posiciones indicadas y aquellos que ya se encontraban presentes en la pantalla simplemente se desplazan. De esta manera se evita tener que instanciar a todos los jugadores en cada iteración. El jugador del cliente es siempre de color blanco y para los demás jugadores se mantiene una estructura que mantiene los nombres de los jugadores y un color asignado al azar. De tal manera de que si el cliente pierde la visión de un jugador y este vuelve a aparecer, tenga el mismo color. El siguiente diagrama ilustra el comportamiento del cliente:

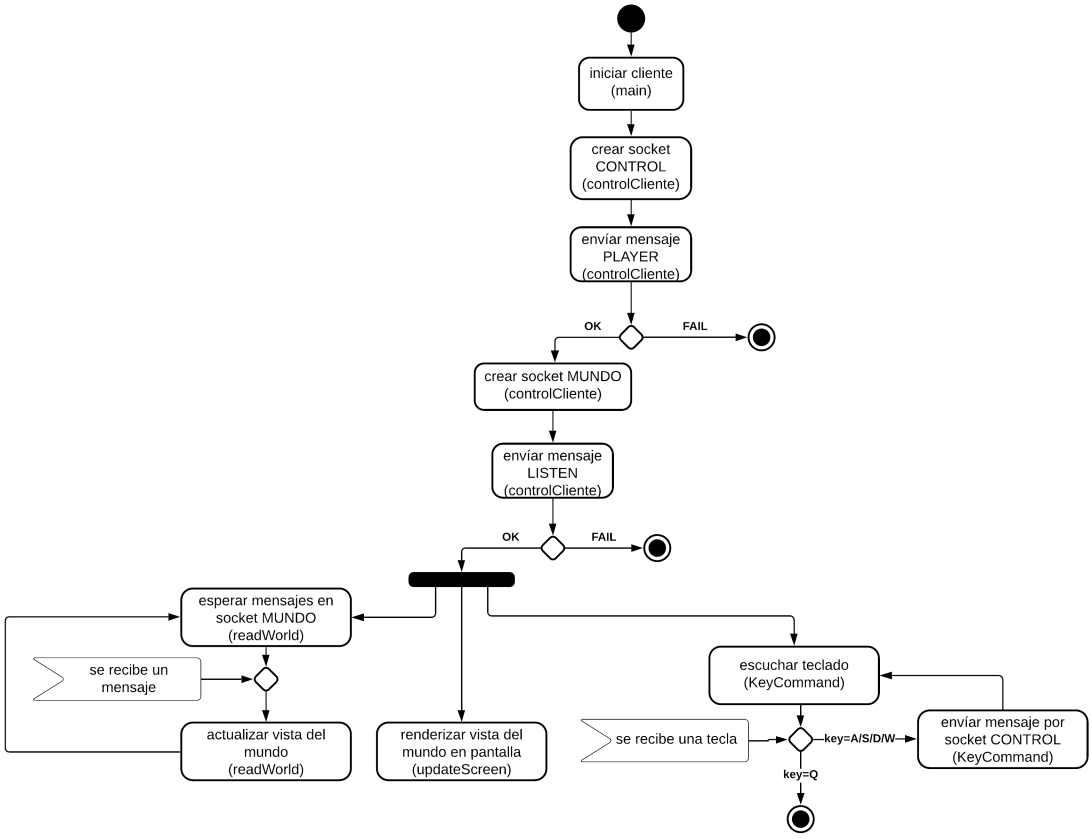

Figure 3: Diagrama de actividad del cliente

3 Script de prueba

Para poder hacer pruebas del sistema de una forma más dinámica y liviana (sin ventana gráfica). Se desarrolló un pequeño script: *testing.py*. El mismo, utiliza las funciones de *cliente.py* para crear "bots" que se mueven en direcciones aleatorias que van cambiando cada 1 segundo. Esto fue de utilidad para testear el correcto funcionamiento interno del sistema y verificar su performance bajo decenas de jugadores interactuando simultáneamente. No notamos perdida de fluidez en el cliente cuando habían menos de 50 jugadores instanciados en el mundo.

4 Conclusión

En líneas generales el grupo quedó muy conforme con el trabajo presentado. Consideramos que los objetivos del obligatorio fueron cumplidos satisfactoriamente.

El trabajo fue de gran utilidad para poner en práctica los conocimientos teóricos adquiridos a lo largo del curso. Un ejemplo de esto es Sockets, los cuales habían sido estudiados en el teórico, pero no puesto en práctica en la realidad. Al tener que trabajar con sockets, utilizando la API de Python, se pudo consolidar mucho más lo aprendido en la teoría.

Adicionalmente para el equipo esta fue la primera vez que desarrollamos una aplicación distribuida de tipo "simulación". Los desafíos que nos encontramos al programar y diseñar una arquitectura que cumpla con los requisitos propuestos lo consideramos una experiencia sumamente enriquecedora que disfrutamos mucho.

5 Especificaciones

La implementación del servidor se encuentra en *servidor.py*. Para ejecutar el cliente se precisan los archivos *cliente.py*, *bg.gif* y *xwing.py*. Estos dos últimos archivos contienen un fondo de pantalla para la ventan y un polígono con forma de nave espacial para la interfaz gráfica del cliente.

servidor.py contiene 8 parámetros configurables, estos y sus valores por defecto son:

- El puerto de CONTROL (2021)
- la velocidad de la simulación (1ms)

- delta tiempo para los broadcasters de MUNDO y el simulador (0.01 s)
- el rango visual de los jugadores (15m)
- la máxima cantidad de conexiones permitidas por host (y la habilitación de dicho control)
- el largo máximo permitido de los nombres de los jugadores (300 chars)
- y el tamaño máximo deseable de los mensajes UDP de MUNDO (Debe ser menor al tamaño máximo de segmento UDP permitido por [2]). (32678 bytes)

cliente.py contiene 5 parámetros configurables:

- La dirección del servidor (loopback 127.0.0.1)
- el puerto de CONTROL (2021)
- el puerto mundo para el cual se desea recibir mensajes (0=asignado por el sistema operativo)
- el tamaño de las tortugas y el tamaño del mundo (para el calculo de su ubicación en la ventana).

References

[1] Python programming language. `https://www.python.org`

[2] Python Sockets Interface. `https://docs.python.org/3/library/socket.html`

[3] Fowler, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Boston: Addison-Wesley, 2004.

[4] Turtle graphics. `https://docs.python.org/3/library/turtle.html`

[5] pynput Package Documentation `https://pynput.readthedocs.io/en/latest`

[6] How to speed up python's 'turtle' function and stop it freezing at the end `https://stackoverflow.com/questions/16119991/how-to-speed-up-pythons-turtle-function-and-stop-it-freezing-at-the-end`