Redes de Computadoras

Obligatorio 2 - 2021

Facultad de Ingeniería
Instituto de Computación
Departamento de Arquitectura de Sistemas

Nota previa - IMPORTANTE

Se debe cumplir íntegramente el "Reglamento del Instituto de Computación ante Instancias de No Individualidad en los Laboratorios", disponible en el EVA.

En particular está prohibido utilizar documentación de otros estudiantes, de otros años, de cualquier índole, o hacer público código a través de cualquier medio (EVA, news, correo, papeles sobre la mesa, etc.).

Introducción

Forma de entrega

Una clara, concisa y descriptiva documentación es clave para comprender el trabajo realizado. La entrega de la tarea consiste en un único archivo obligatorio2GrupoGG.tar.gz que deberá contener los siguientes archivos:

- Un documento llamado Obligatorio2GrupoGG.pdf donde se documente todo lo solicitado en la tarea. GG es el número del grupo. La documentación deberá describir claramente su solución, las decisiones tomadas, los problemas encontrados y posibles mejoras.
- El código fuente del programa (**en lenguaje Python**) e instrucciones claras de cómo ejecutar el sistema.

La entrega se realizará en el sitio del curso, en la plataforma EVA.

Fecha de entrega

Los trabajos deberán ser entregados **antes del 10/10/2021 a las 23:30 horas**. No se aceptará ningún trabajo pasada la citada fecha y hora. En particular, no se aceptarán trabajos enviados por e-mail a los docentes del curso.

Objetivo del Trabajo

Aplicar los conceptos teóricos de capas de aplicación y transporte, la utilización de la API de sockets TCP y UDP, y la arquitectura de aplicaciones cliente-servidor [1].

Descripción general del problema

Se desea implementar un juego MMO (*Massively multiplayer online game*).

Problema a resolver

Se propone desarrollar un juego en línea, siguiendo una arquitectura cliente-servidor. Se toman medidas especiales para combatir la posibilidad de hacer trampas mediante *bots*. En particular, los clientes solo tienen acceso a la información del mundo que el servidor computa que le debe estar disponible.

1. Servidor

El servidor contiene la simulación de un mundo compartido, donde los jugadores coexisten. Un nuevo jugador es instanciado cuando se conecta un cliente, y se destruye cuando el cliente se desconecta.
El mundo simulado consiste en un mapa de 100 metros * 100 metros, en el que se mueven los jugadores. Las coordenadas son de tipo punto flotante. Los jugadores se mueven sin detenerse a una velocidad constante de 1m/s en una de las cuatro direcciones N, S, E y O. Los jugadores se consideran puntos sin masa, y no se prevé que colisionen ni que interactúen de ninguna forma. Se recomienda procesar la simulación del mundo a una velocidad aproximada de 100 actualizaciones por segundo (ver Anexo 1).
La dirección en la que se mueve será controlada desde el cliente asociado al jugador (usando el protocolo CONTROL).
Cada jugador tiene un rango visual de 15 metros, configurables al levantar el servidor. El servidor computará cuáles de los co-participantes están dentro del rango visual del jugador y le enviará una lista de jugadores visibles con sus posiciones (usando el protocolo MUNDO).

2. Clientes

Los clientes se conectan para controlar un avatar de un jugador (enviando comandos por el protocolo CONTROL). Los comandos se leen del teclado de forma interactiva, por ejemplo los cursores o las teclas w-a-s-d (ver Anexo 1).
Los clientes continuamente recibirán desde el servidor las coordenadas del jugador y los co-participantes que sean visibles (usando el protocolo MUNDO), y los visualizarán. Para visualizarlos se proporciona una función [2] que dibuja a cada avatar en pantalla usando la biblioteca *turtle* [3]. Además se sugiere imprimir en la consola la lista de jugadores y sus coordenadas.

3. Protocolos

Se definen dos protocolos. El protocolo CONTROL se utiliza para gestionar la sesión del jugador y enviar comandos de control. El protocolo MUNDO se usa para enviar el estado del mundo del servidor hacia los clientes.

3.1 Protocolo: CONTROL.

El protocolo de CONTROL es usado por los clientes para establecer la sesión de juego y para controlar a su avatar. Utiliza el transporte TCP conectándose al puerto 2021 del servidor. El protocolo consiste de comandos separados por el carácter **\n** (línea nueva). Los comandos son siempre iniciados desde el cliente, y algunos comandos pueden disparar una respuesta del servidor, según se detalla a continuación.

Los comandos soportados son:

3.1.1 Login

Se usa para instanciar un jugador. Tiene el siguiente formato:

```
PLAYER <username>\n
```

Este debe ser el primer comando que recibe el servidor. Causará que se instancie un nuevo jugador con el nombre indicado, en un lugar aleatorio del mapa. El nombre debe estar compuesto de caracteres alfanuméricos, sin espacios en blanco. Este jugador existirá hasta que la sesión TCP se cierre.
El servidor responderá

```
OK\n
```

si el jugador fue instanciado exitosamente, o

```
FAIL <errormessage>\n
```

en caso de algún fallo (por ejemplo, que el nombre esté ocupado). En caso de fallo el servidor cerrará la conexión.

3.1.2 Dirección de cliente

Se usa para indicarle al servidor el puerto usado por el cliente para recibir los mensajes del protocolo MUNDO:

```
LISTEN <portnumber>\n
```

El servidor responderá

```
OK\n
```

si el jugador fue configurado exitosamente, o

```
FAIL <errormessage>\n
```

en caso de algún fallo. En caso de fallo el servidor cerrará la conexión.

3.1.2 Comandos de control

Se usan para indicar la dirección en la que debe empezar a moverse el jugador. Los comandos posibles son:

```
GO N\n
GO S\n
GO E\n
GO W\n
```

El servidor no responde a estos mensajes. Será responsabilidad del emisor no saturar el enlace con estos mensajes: solo se deben emitir cuando se realiza un cambio de dirección de movimiento.

3.2 Protocolo: MUNDO

Este protocolo se usa para hacer *streaming* del estado del mundo para cada jugador. Utiliza el transporte UDP. El flujo de datos es unidireccional del servidor a los clientes. Cada cliente escucha en la dirección <IP, PUERTO> donde IP es la misma dirección IP desde la que se estableció la conexión de CONTROL, y PUERTO es el puerto presentado mediante el comando LISTEN en la sesión de CONTROL.
El servidor envía a cada cliente mensajes conteniendo los jugadores visibles para ese cliente. Los mensajes tienen el formato:

```
WORLD <timestamp>\n
PLAYER <x> <y> <direction>\n
<username1> <x1> <y1> <direction1>\n
<username2> <x2> <y2> <direction2>\n
<username3> <x3> <y3> <direction3>\n
...
```

El campo <*timestamp*> contiene el tiempo de la simulación en milisegundos.
La linea PLAYER es obligatoria, y contiene las coordenadas del jugador y su dirección (‘`N`’, ‘`S`’, ‘`E`’ o ‘`W`’).
Las líneas siguientes son opcionales, y contienen datos de jugadores dentro del radio visual de PLAYER.
La forma de repartir jugadores en mensajes WORLD y la política de su emisión es dependiente de la implementación del servidor. Se considera que la velocidad óptima de refresco de los datos del mundo para los clientes es de 10 FPS o actualizaciones por segundo (ver Anexo 1).

4. Se pide

a) Diseñe y documente la arquitectura de la aplicación descrita anteriormente.

b) Implemente en el lenguaje Python, utilizando las primitivas de la API de sockets correspondiente [4][5][6], las aplicaciones cliente y servidor descritas anteriormente.

5. Observaciones:

- Se aceptará el uso de bibliotecas de estructuras de datos, interacción con el teclado, hilos, etc.

Referencias y Bibliografía Recomendada

[1] Kurose, James, y Keith W. Ross. *Redes de computadoras*. Vol. 7. Pearson Educación, 2017
[2] Código para visualización - https://eva.fing.edu.uy/mod/resource/view.php?id=147659
[3] turtle — Turtle graphics. https://docs.python.org/3/library/turtle.html
[4] Comer, Douglas, *Computer Networks and Internets*, 5th Edition.
[5] Linux man-pages. https://www.kernel.org/doc/man-pages/
[6] Python Sockets Interface. https://docs.python.org/3/library/socket.html

Anexo 1

Se sugiere el siguiente pseudocódigo para implementar el simulador en el servidor:

```
type Player: {string name, float x, float y, char direction}
players = List of Player

v = 1 m/s
dt_sim = 0.01s

thread.new( function()
  while true do
    lock(players) # controlar acceso desde varios hilos
    for each Player p in players
      if p.direction == 'N'
        y -= v*dt_sim
      elseif p.direction == 'S'
        p.y += v*dt_sim
      elseif p.direction == 'E'
        p.x += v*dt_sim
      elseif p.direction == 'N'
        p.y -= v*dt_sim
      clip( p.x, 0.0, 100.0 ) # si se sale del mapa llevar al borde
      clip( p.y, 0.0, 100.0 )

    unlock(players)
    sleep(dt_sim)
  end
end)
```

El envío de los datos en el servidor se podría estructurar de la siguiente manera:

```
dt_send = 0.1s
R = 15m

thread.new( function()
  while true do
    lock(players)
    for each Player p in players
      neighbors = FindCloserThan(p, players, 15) # ListOf Player
      message = BuildMessage(p, os.time(), neighbors)
      send_message(p, message)
    unlock(players)
    sleep(dt_send)
  end
end)
```

Para el manejo del teclado en el cliente, se sugiere evaluar las bibliotecas *keyboard* o *pynput*. Por ejemplo https://www.delftstack.com/howto/python/python-detect-keypress/.