

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS



# Redes de Computadoras

## *Pokemon GO*

Profesor: Paulo Santiago de Jesús Contreras Flores

Ayudante: Ulises Manuel Cárdenas

Ayudante: Ismael Andrade Canales

*Barajas Figueroa José de Jesús*

*314341015*

*Monreal Gamboa Francisco Manuel*

*314036126*

*Ramirez Garcia Diana Isabel*

*314127529*

Proyecto correspondientes al curso de *Redes de Computadoras*, impartido en el semestre  
2020-1 por el profesor:

PAULO SANTIAGO DE JESÚS CONTRERAS FLORES

2020-1

# Índice

<b>1</b>	<b>Objetivo</b>	<b>2</b>
<b>2</b>	<b>Diseño del protocolo</b>	<b>4</b>
2.1	Socket	4
<b>3</b>	<b>Uso</b>	<b>4</b>
3.1	make all	6
3.2	sudo make build_PokemonGo	7
3.3	Servidor	8
3.4	Cliente	8
3.5	TCP Dump	8
<b>4</b>	<b>Lista de funciones usadas en la programación</b>	<b>10</b>
4.1	Funciones de bibliotecas de Python	10
4.2	Cliente	10
4.3	Servidor	11

# 1. Objetivo

En pocas ocasiones podemos afirmar que la ficción supera a la realidad, esta es una de esas ocasiones que ocurren una o dos veces en la vida, PokemonGo lo hace posible trayendo una nueva experiencia de juego con estos enigmáticos personajes que nos han acompañado desde ya hace varias décadas, en esta ocasión salimos de un mundo bidimensional para traerlos a un entorno completamente interactivo.



Figura 1: Monitoreo de Pokemon, su nivel, su fuerza, su defensa, sus habilidades; Cada rubro completamente desglosado



Figura 2: Usando tecnología de realidad aumentado, los Pokemons están en todos lados



Figura 3: Debes de localizar un Pokemon y emprender la aventura de buscarlos

PokemonGo sale del estándar de videojuego siendo ahora fundamental tener que salir al mundo exterior e interactuar con el ambiente haciendo realista la experiencia Pokemon.

## 2. Diseño del protocolo

### 2.1. Socket

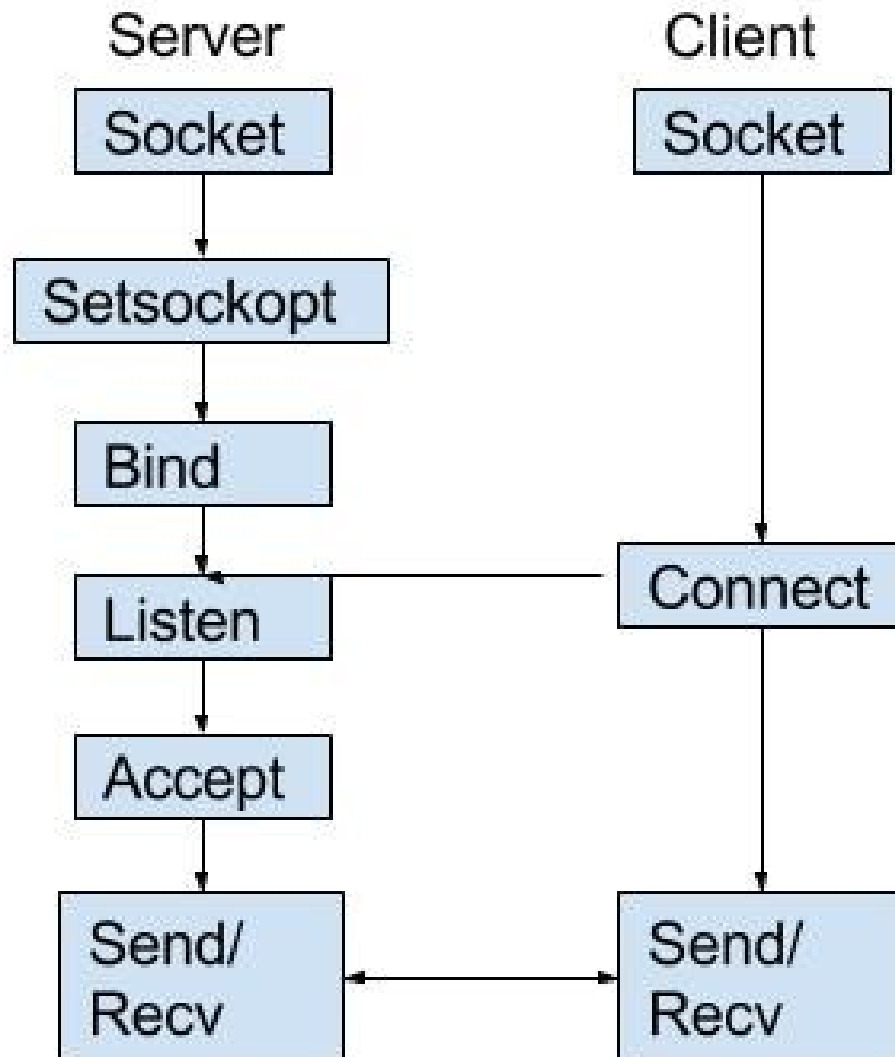


Figura 4: Diagrama de secuencia de una conexión entre un cliente y un servidor mediante las funciones de la API socket de Python, usando el protocolo TCP, este es un esquema general a grandes rasgos, lo utilizamos para darnos una primer idea de como funciona en la práctica un socket

## 3. Uso

<https://github.com/jebarfig21/PokemonGo>, en esta liga se encuentra el repositorio de

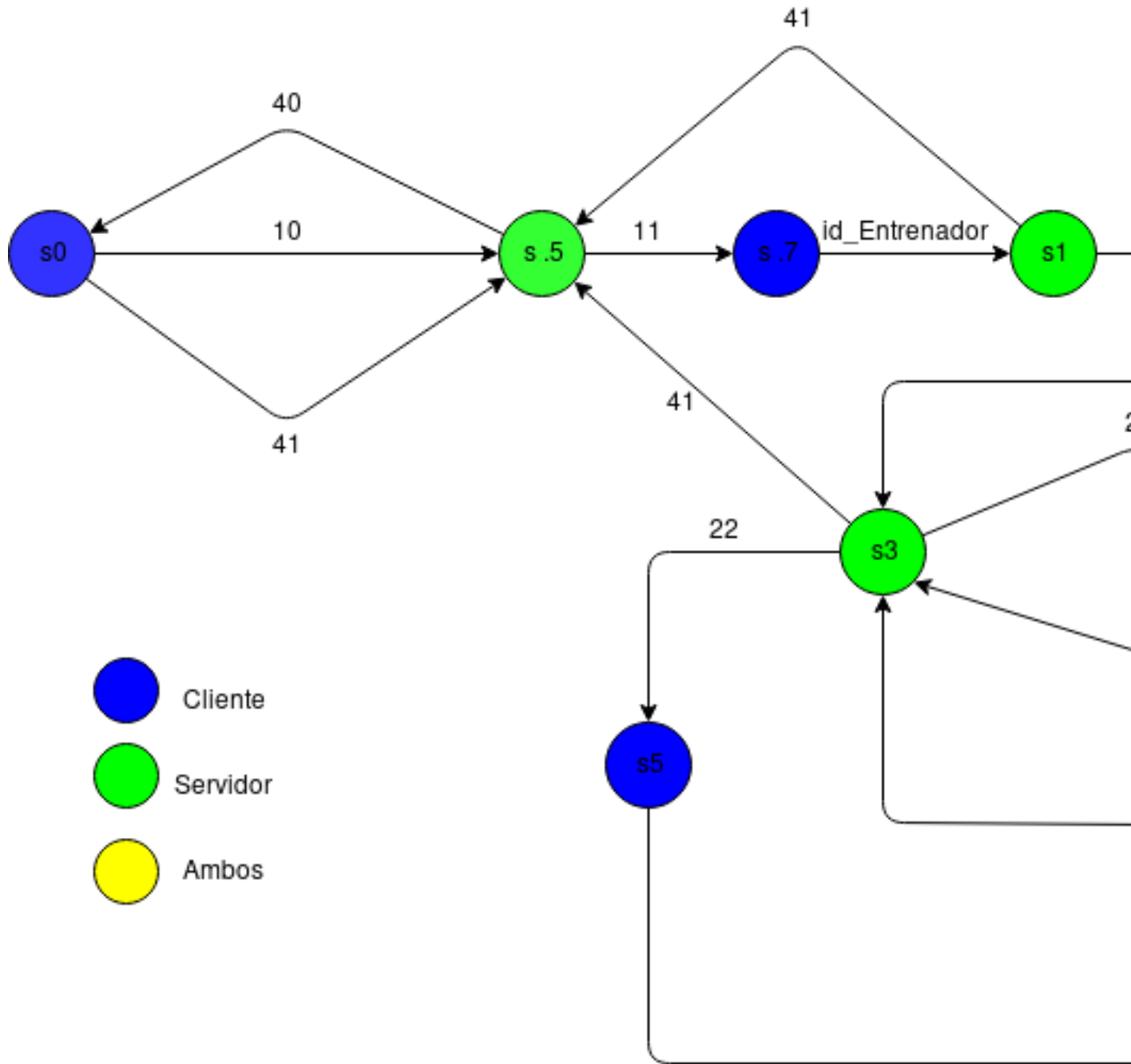


Figura 5: FSM que usamos nosotros, es muy parecido al propuesto en los requerimientos, sin embargo, le añadimos 3 códigos, 2 para errores(41, 40) y unos para solicitar entrenador(11), así como la transición en la que se solicita el id del entrenador

Número	Acción	Lado
10	Solicitar captura	cliente
11	Solicitar entrenador	servidor
20	Preguntar captura	servidor
21	Capturar de nuevo	servidor
22	Enviar Pokemon	servidor
23	Intentos agotados	servidor
30	Si	ambos
31	No	ambos
32	Terminar conexión	ambos
40	Error de conexión	Servidor
41	Error de código	Servidor

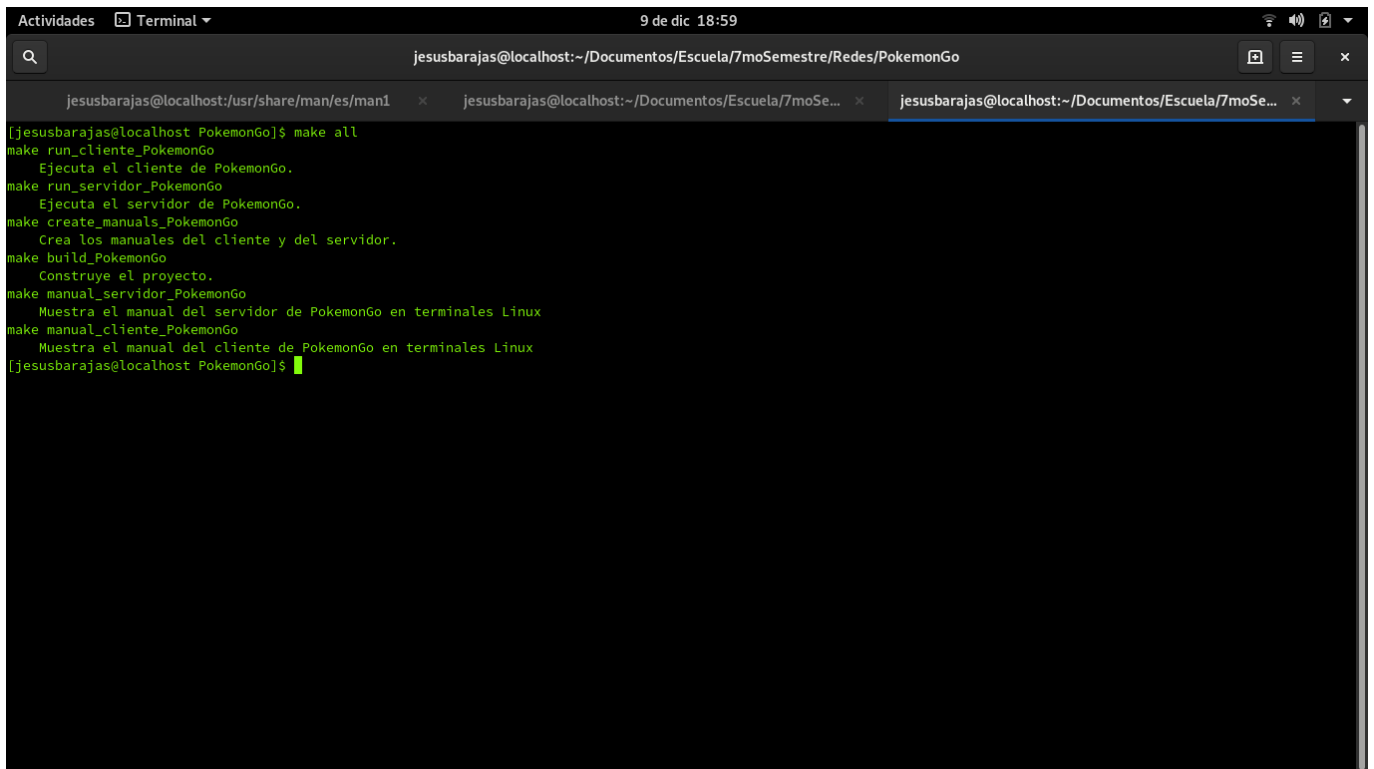
Cuadro 1: Descripción de los códigos implementados en el desarrollo del proyecto, los cuales corresponden con el diagrama planteado en inicio, sumando las adecuaciones necesarias para regresar códigos de error

github donde tenemos nuestro proyecto

El programa funciona mediante la secuencia de ejecución *make*, las cuales son configurables en el archivo *Makefile*, la documentación se encuentra en la carpeta doc y fue hecha mediante sphinx, sin embargo la documentación se puede consultar en el archivo index.html que se encuentra en el directorio raíz, en la carpeta man se encuentran los manuales de linux, el archivo servidor.py tiene el programa del servidor y el programa cliente.py el del cliente. También incluimos una captura de tráfico de tcp dump, usando otra computadora en la red local como cliente y otra como servidor, lo obtenido se puede revisar en el archivo *captura1.txt*. Los Pokemons que han sido capturados se encuentran en la carpeta pokemons\_capturados que se encuentra en el directorio raíz

### 3.1. make all

Con esta secuencia podremos ver todos los comandos disponibles para *make*, decidimos poner las instrucciones de servidor y cliente en un mismo archivo para que cualquier máquina pueda ser o cliente o servidor.

A screenshot of a terminal window with a dark background. The window title bar shows 'Actividades' and 'Terminal'. The address bar at the top displays 'jesusbarajas@localhost:~/Documentos/Escuela/7moSemestre/Redes/PokemonGo'. The terminal content shows the following commands and their outputs:

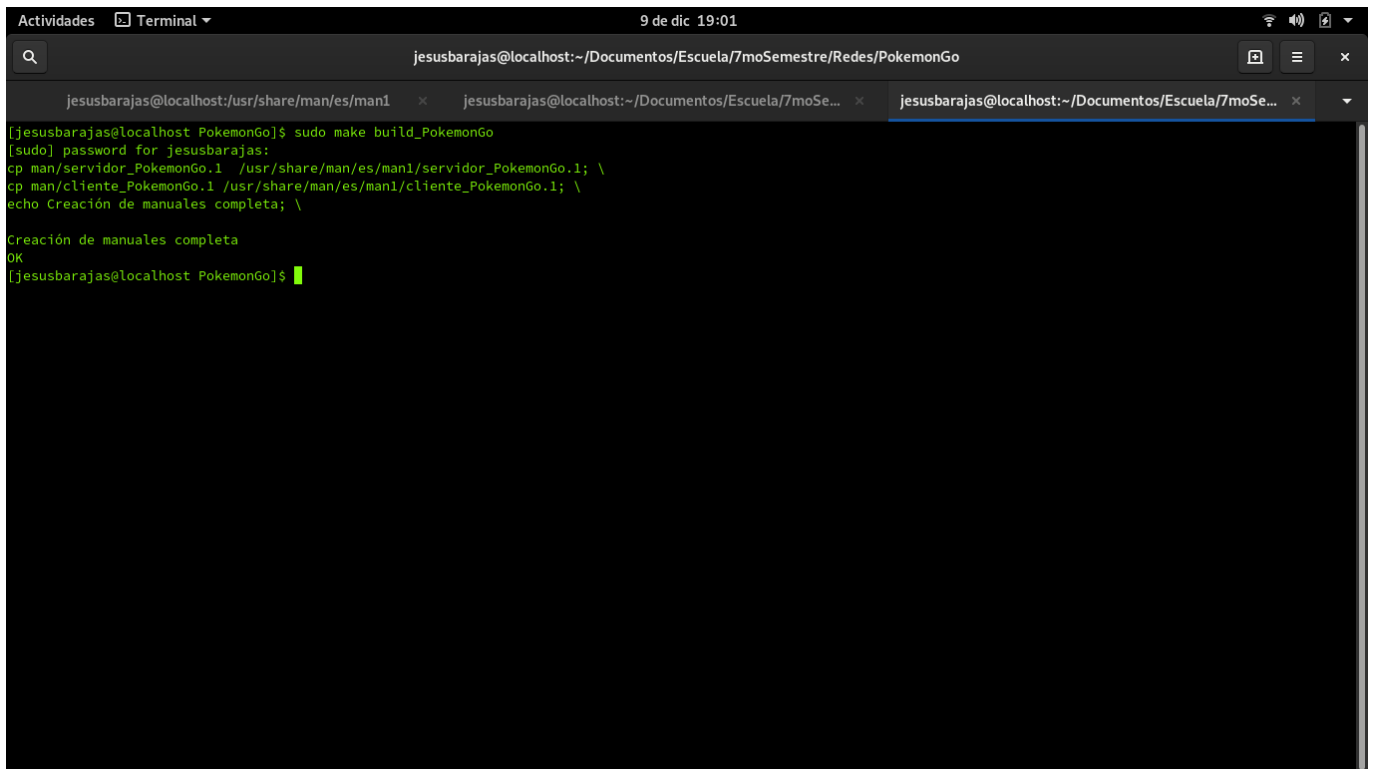
```
[jesusbarajas@localhost PokemonGo]$ make all
make run_cliente_PokemonGo
Ejecuta el cliente de PokemonGo.
make run_servidor_PokemonGo
Ejecuta el servidor de PokemonGo.
make create_manuales_PokemonGo
Crea los manuales del cliente y del servidor.
make build_PokemonGo
Construye el proyecto.
make manual_servidor_PokemonGo
Muestra el manual del servidor de PokemonGo en terminales Linux
make manual_cliente_PokemonGo
Muestra el manual del cliente de PokemonGo en terminales Linux
[jesusbarajas@localhost PokemonGo]$
```

Figura 6: Uso del comando make all

### 3.2. sudo make build\_PokemonGo

Antes que nada es importante correr este comando para poder crear los manuales, cabe aclarar que el comando necesita ser ejecutado como súper usuario, este comando funciona con varias distribuciones Linux, en particular fue probada en Fedora 30, sin embargo no sabemos como se comporte en otras distribuciones.





```
Actividades Terminal 9 de dic 19:01
jesusbarajas@localhost:~/Documentos/Escuela/7moSemestre/Redes/PokemonGo

jesusbarajas@localhost:usr/share/man/es/man1 x jesusbarajas@localhost:~/Documentos/Escuela/7moSe... x jesusbarajas@localhost:~/Documentos/Escuela/7moSe... x
[jesusbarajas@localhost PokemonGo]$ sudo make build_PokemonGo
[sudo] password for jesusbarajas:
cp man/servidor_PokemonGo.1 /usr/share/man/es/man1/servidor_PokemonGo.1; \
cp man/cliente_PokemonGo.1 /usr/share/man/es/man1/cliente_PokemonGo.1; \
echo Creación de manuales completa; \

Creación de manuales completa
OK
[jesusbarajas@localhost PokemonGo]$
```

Figura 7: Uso del comando make build\_PokemonGo

### 3.3. Servidor

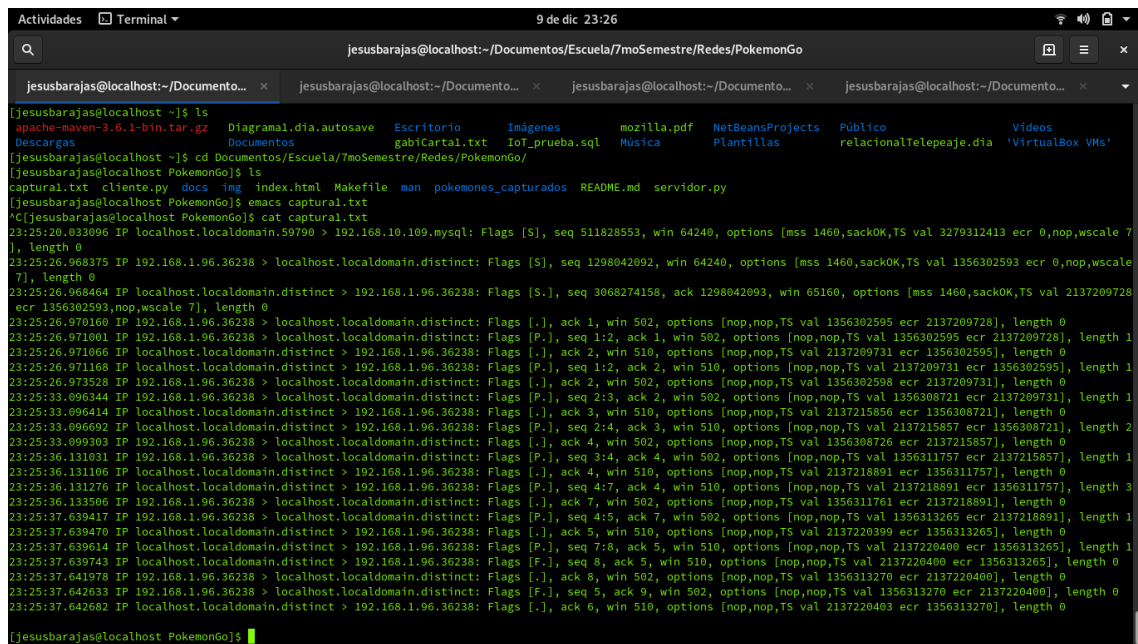
- **make run\_servidor\_PokemonGo** : Con este comando corremos el programa del lado del servidor.

### 3.4. Cliente

- **make run\_cliente\_PokemonGo** : Usamos este comando para iniciar la ejecución del programa del lado del cliente
- Dirección IP : Posteriormente se va a solicitar la dirección IP donde esta el servidor, se debe ingresar esta dirección IP
- Id Usuario : Ahora se va a solicitar que se ingrese el número de identificación del entrenador, por ahora solo existen 3 usuarios, 1, 2 y 3.

### 3.5. TCP Dump

[H]



```
Actividades Terminal 9 de dic 23:26
jesusbarajas@localhost:~/Documentos/Escuela/7moSemestre/Redes/PokemonGo

jesusbarajas@localhost:~/Documento... x jesusbarajas@localhost:~/Documento... x jesusbarajas@localhost:~/Documento... x jesusbarajas@localhost:~/Documento... x

[jesusbarajas@localhost ~]$ ls
apache-maven-3.6.1-bin.tar.gz  Diagrama1.dia.autosave  Escritorio  Imágenes  mozilla.pdf  NetBeansProjects  Público  Videos
Descargas  Documentos  gabiCartal.txt  IoI_prueba.sql  Musica  Plantillas  relacionalTelepeaje.dia  'VirtualBox VMs'

[jesusbarajas@localhost ~]$ cd Documentos/Escuela/7moSemestre/Redes/PokemonGo/
[jesusbarajas@localhost PokemonGo]$ ls
captural.txt  cliente.py  docs  img  index.html  Makefile  man  pokemones_capturados  README.md  servidor.py
[jesusbarajas@localhost PokemonGo]$ emacs captural.txt
^C[jesusbarajas@localhost PokemonGo]$ cat captural.txt
23:25:20.833096 IP localhost.localdomain.59790 > 192.168.10.109.mysql: Flags [S], seq 511828553, win 64240, options [mss 1460,sackOK,TS val 32793112413 ecr 0,nop,wscale 7], length 0
23:25:26.968375 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [S], seq 1298042092, win 64240, options [mss 1460,sackOK,TS val 1356302593 ecr 0,nop,wscale 7], length 0
23:25:26.968464 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [S.], seq 3068274158, ack 1298042093, win 65160, options [mss 1460,sackOK,TS val 2137209728 ecr 1356302593,nop,wscale 7], length 0
23:25:26.970160 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [S.], seq 3068274158, ack 1298042093, win 65160, options [mss 1460,sackOK,TS val 2137209728 ecr 1356302593,nop,wscale 7], length 0
23:25:26.971001 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 1:2, ack 1, win 502, options [nop,nop,TS val 1356302595 ecr 2137209728], length 1
23:25:26.971066 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 2, win 510, options [nop,nop,TS val 2137209731 ecr 1356302595], length 0
23:25:26.971168 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 1:2, ack 2, win 510, options [nop,nop,TS val 2137209731 ecr 1356302595], length 1
23:25:26.973528 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 2, win 502, options [nop,nop,TS val 1356302598 ecr 2137209731], length 0
23:25:33.896044 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 2:3, ack 2, win 502, options [nop,nop,TS val 1356308721 ecr 2137209731], length 1
23:25:33.896414 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 3, win 510, options [nop,nop,TS val 2137215856 ecr 1356308721], length 0
23:25:33.896592 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 2:4, ack 3, win 510, options [nop,nop,TS val 2137215857 ecr 1356308721], length 2
23:25:33.896903 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 4, win 502, options [nop,nop,TS val 1356308726 ecr 2137215857], length 0
23:25:36.131031 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 3:4, ack 4, win 502, options [nop,nop,TS val 1356311757 ecr 2137215857], length 1
23:25:36.131106 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 4, win 510, options [nop,nop,TS val 2137218891 ecr 1356311757], length 0
23:25:36.131276 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 4:7, ack 4, win 510, options [nop,nop,TS val 2137218891 ecr 1356311757], length 3
23:25:36.133506 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 7, win 502, options [nop,nop,TS val 1356311761 ecr 2137218891], length 0
23:25:37.639417 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 4:5, ack 7, win 502, options [nop,nop,TS val 1356313265 ecr 2137218891], length 1
23:25:37.639470 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 5, win 510, options [nop,nop,TS val 2137220399 ecr 1356313265], length 0
23:25:37.639614 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 7:8, ack 5, win 510, options [nop,nop,TS val 2137220400 ecr 1356313265], length 1
23:25:37.639743 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [F.], seq 8, ack 5, win 510, options [nop,nop,TS val 2137220400 ecr 1356313265], length 0
23:25:37.641978 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [P.], seq 8, win 502, options [nop,nop,TS val 1356313270 ecr 2137220400], length 0
23:25:37.642633 IP 192.168.1.96.36238 > localhost.localdomain.distinct: Flags [F.], seq 5, ack 9, win 502, options [nop,nop,TS val 1356313270 ecr 2137220400], length 0
23:25:37.642682 IP localhost.localdomain.distinct > 192.168.1.96.36238: Flags [P.], seq 6, win 510, options [nop,nop,TS val 2137220403 ecr 1356313270], length 0

[jesusbarajas@localhost PokemonGo]$
```

Figura 8: Captura obtenida, en este intento no se logro capturar al Pokemon, era un eve, fueron 2 ataques, eve fue fuerte y logro escapar. Cabe aclarar que esta captura de tráfico la hicimos con el programa tcpdump y no con wireshark ya que la computadora donde hicimos las pruebas no abre la interfaz de wireshark, se usó el siguiente comando *sudo tcpdump -i wlp2s0 tcp >captura1.txt*

## 4. Lista de funciones usadas en la programación

### 4.1. Funciones de bibliotecas de Python

- Socket Supongamos que tenemos un objeto de esta clase llamado
  - `socket(familia, protocolo)` : Con el comando `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` iniciamos la instancia del objeto, el primer argumento recibe la familia de direcciones, `AF_INET` corresponde a la familia del protocolo IPv4, mientras que el segundo argumento `SOCK_STREAM`, corresponde al protocolo que se usara en la capa de transporte, que en este caso sera TCP. Este método funciona igual para el servidor como para el cliente, es necesario iniciar el objeto con este método en ambos lados.
  - `bind(host, puerto)` : Son `socket.bind(host, puerto)` solicitamos abrir la conexión en el host y en el puerto que se solicita como argumento. Este método depende de la familia que haya sido seleccionada en el método *socket*, como usamos IPv4, es suficiente con esos 2 argumentos. Este método se ejecuta desde el lado del servidor.
  - `listen()` : con el método `socket.listen()` le diremos a nuestro objeto que sea capaz de aceptar conexiones, coloquialmente se conoce como levantar el servidor en modo escucha. Este método se ejecuta desde el lado del servidor.
  - `accept()` : Es para aceptar la comunicación que viene de algún cliente, es importante aclarar que a partir de este momento el servidor ya tiene conexión con el cliente, este método pertenece al servidor.
  - `connect(host, puerto)` : Con este método el cliente solicita conectarse a un host por medio de un puerto.
  - `send(message)` : Con este método el cliente manda un mensaje.
  - `recv()` Este método guarda la respuesta del cliente al mensaje enviado.
  - `close()` Este método es necesario en ambos lados

### 4.2. Cliente

- `conectarServidor` : Función que manda la solicitud de conexión a un servidor.

- `obtenerEntrenador` : Función que solicita la autenticación de un entrenador o usuario, en caso de no existir solicita de nuevo intentar autenticarse
- `bytes_converter` : Función auxiliar que convierte elementos de tipo bytes a tipo entero
- `capture_pokemon` : Función que ofrece un Pokemon a capturar, si se decide empezar a capturar se tienen hasta 10 intentos(se escogen al azar entre 2 a 10 intentos), al pasar estos intentos el Pokemon escapa, el mejor escenario es capturar al pokemon, si esto pasa se guarda en pokemon capturados.

### 4.3. Servidor

- `pedirEntrenador` : Se solicita el id del entrenador, como utilizamos un diccionario en python la autenticación la hicimos directo a buscar en el diccionario si se encuentra, si el número es mayor que el número máximo de elemento, simplemente rechazamos, por ahora el número de límite lo pusimos manualmente
- `comprueba10` : Método que comprueba que el código de solicitud de conexión haya sido 10
- `escucha` : Método que abre nuestro socket del lado del servidor en modo escucha latente desde que se ejecuta el programa hasta que se termina, es importante aclarar que se aceptan conexión concurrentes gracias al manejo de hilos
- `conexiónCliente` : Método principal donde se va a interactuar con el cliente, este método va a corroborar los código correspondientes, también a a ofrecer los Pokemon que tiene disponibles, también esta al tanto de los timeouts. Esta función utiliza los métodos siguientes para interactuar con el cliente y los Pokemon
- `img_bytes` : Función auxiliar que convierte el tamaño de una imagen a bytes.
- `mandar_pokemon` : Esta función permite obtener un Pokemon y ofrecerlo al cliente.
- Es la función con la que vamos a intentar hacer que el cliente obtenga o no el Pokemon, también revisa los intentos, y si el cliente quiere seguir intentando capturar o no.

## Referencias

- [1] Nathan Jennings , Socket Programming in Python (Guide), RealPython <https://realpython.com/python-sockets/>. Consultado el día 7 de Diciembre de 2019
- [2] Nathan Jennings , Socket Programming in Python (Guide), RealPython <https://www.linux-party.com/54-programacion/6968-crear-paginas-man-del-manual-para-linux>. Consultado el día 9 de Diciembre de 2019
- [3] Nathan Jennings , Socket Programming in Python (Guide), RealPython <https://krzysztofzuraw.com/blog/2016/makefiles-in-python-projects.html>. Consultado el día 9 de Diciembre de 2019