Taller 1

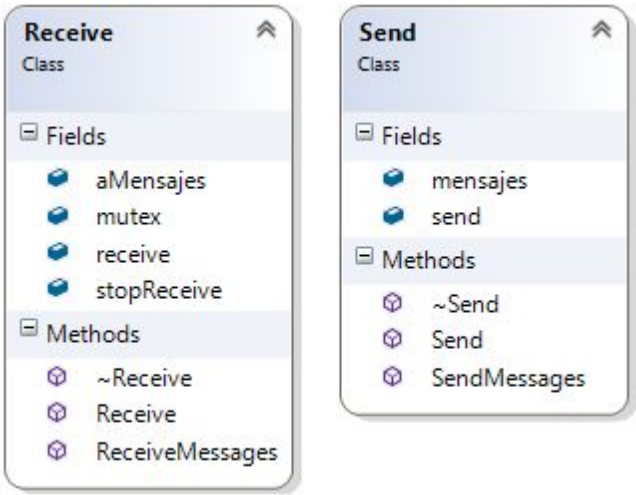
Poner en marcha el proyecto

1. Abrir proyecto
2. Escribir la IP del servidor

```
16      sf::IpAddress ip = sf::IpAddress::IpAddress("213.148.194.122");
```

3. Ejecutar proyecto (F5)
4. Escribir ‘s’ para servidor o ‘c’ para cliente y presionar Entrar
5. Para salir, presionar Escapar

Diagrama de clases



Sockets Send y Receive para cliente y servidor

En el source.cpp

```
17      sf::TcpSocket *send = new sf::TcpSocket;
18      sf::TcpSocket *receive = new sf::TcpSocket;
```

Thread para la recepción de mensajes

En el source.cpp

```
109     sf::Thread Threceive(&Receive::ReceiveMessages, &receiver);
110     Threceive.launch();
```

Lista de mensajes

En el source.cpp

```
72      std::vector<std::string> aMensajes;
73      receiver.aMensajes = &aMensajes;
```

La clase recepción tiene un puntero vector<string> para acceder a la lista de mensajes.

Recibo de mensajes

En el Receive.cpp.

```
15  void Receive::ReceiveMessages()
16  {
17      char data[1500];
18      std::size_t received;
19      std::string mensaje = "";
20      while (!stopReceive) {
21          sf::Socket::Status status = receive->receive(data, 1500, received);
22          data[received] = '\0';
23
24          mensaje = data;
25          mutex->lock();
26          aMensajes->push_back(mensaje);// guarda mensaje a la llista de mensajes
27          mutex->unlock();
28      }
29  }
30
```

Envío de mensajes

En el Send.cpp.

```
15  void Send::SendMessages()
16  {
17      if (send->send(mensajes->c_str(),mensajes->size()) != sf::Socket::Done)
18      {
19          std::cout << "Error al enviar" << mensajes << std::endl;
20      }
21  }
```

Impresion mensajes por pantalla

En el Source.cpp

```
157         window.draw(separator);
158         for (size_t i = 0; i < aMensajes.size(); i++)
159         {
160             std::string chatting = aMensajes[i];
161             chattingText.setPosition(sf::Vector2f(0, 20 * i));
162             chattingText.setString(chatting);
163             window.draw(chattingText);
164         }
165         std::string mensaje_ = mensaje + "_";
166         text.setString(mensaje_);
167         window.draw(text);
168         window.display();
169         window.clear();
```

Mutex

En el source.cpp se crea.

```
12  int main()
13  {
14      sf::Mutex mutex;
```

En el send del source.cpp se bloquea antes de poner el mensaje enviado en la lista y se desbloquea justo después

```
134             else if (evento.key.code == sf::Keyboard::Return)
135             {
136
137                 sender.SendMessages(); // envia mensaje
138                 mutex.lock();
139                 aMensajes.push_back(mensaje);
140                 mutex.unlock();
```

En el receive del Receive.cpp se bloquea antes de poner el mensaje recibido en la lista y se desbloquea justo después. El objeto receive tiene un puntero al mutex del source.cpp.

```
25          mutex->lock();
26          aMensajes->push_back(mensaje);
27          mutex->unlock();
```

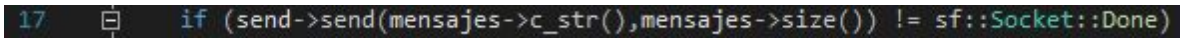
Desconexión de los sockets

En el Source.cpp

```
172     send->disconnect();
173     receive->disconnect();
174     Threceive.terminate();
175     return 0;
```

Tipo de datos

Usamos RAW data para acostumbrarnos para cuando usemos UDP



Postmortem

La mayor dificultad que hemos tenido ha sido a la hora de crear los threads. Al principio queríamos crear un thread con la función receive del source.cpp, pero como no podíamos pasarle dos parámetros (el socket y la lista de mensajes), pensamos en poner toda la ejecución dentro de una clase Chat. La clase Chat no nos funcionó ya que no supimos crear un thread que funcionase bien dentro de la clase, y al final optamos por crear una clase Send y una clase Receive y ejecutar un thread de la función receive de un objeto Receive en el source.cpp.

Ha sido una buena práctica para empezar ya que hemos refrescado lo que aprendimos en sistemas operativos y redes.