Taller 2

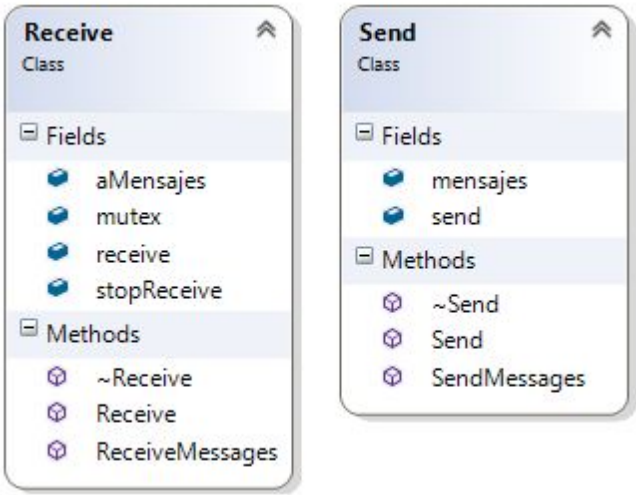
Poner en marcha el proyecto

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

```
16      sf::TcpSocket socket;
```

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.

```
13      char data[1500];
14      std::size_t received;
15      std::string mensaje = "";
16      sf::Socket::Status status = receive->receive(data, 1500, received);
17      data[received] = '\0';
18
19      if (status == sf::Socket::Done) {
20          mensaje = data;
21          aMensajes->push_back(mensaje);// guarda mensaje a la llista de mensajes
22
23          if (aMensajes->size() > 25)
24          {
25              aMensajes->erase(aMensajes->begin(), aMensajes->begin() + 1);
26          }
27      }
```

Envío de mensajes

En el Send.cpp.

```
13      size_t sent;
14      sf::Socket::Status status;
15      do {
16          status = send->send(mensajes->c_str(), mensajes->size(), sent);
17          if (status != sf::Socket::Done)
18          {
19              std::cout << "Error al enviar" << mensajes << std::endl;
20          }
21      } while (status != sf::Socket::Done);
22
```

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();
```

Blocking

En el source.cpp que comparten cliente y servidor.

```
92      socket.setBlocking(false);
```

Al crear el socket, está por defecto en blocking y lo usamos para hacer el connect/accept. Justo antes de empezar el bucle de send/receive, lo ponemos en nonBlocking hasta el final del programa.

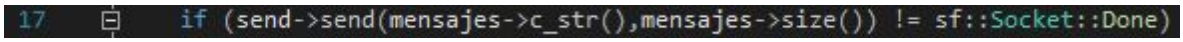
Desconexión de los sockets

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

En este taller 2, la mayor dificultad ha sido la de readaptar el código usado en el Taller 1. Tuvimos algunos problemas ya que aparecieron diferentes problemas que no supimos solucionar y tuvimos que repetir el código hasta 2 veces, después surgieron errores derivados de estas múltiples copias.

Aunque parezca extraño hemos tenido más problemas en readaptar el código más que no en hacer esta práctica. Ya que el sistema de Nonblocking es mucho más simple y más rápido de implementar. Como pequeña dificultad al crear las sockets non blocking fue la de utilizar la función sobrecargada del send, ya que con el sistema anterior no era obligatorio usarla.