Taller 6 – UDP (Optimización de datos)

(He unido algunas peticiones del taller 5 con el taller 6 para que se entendiese mejor el proyecto, debido a que empecé por el taller 6 y con los requisitos de entrega del taller 6 quizás no se entendía muy bien todo)

¿Cómo poner en marcha el proyecto?

**Dentro del .zip se encuentra el archivo de la solución y dentro de la carpeta x64 están los ejecutables GameClient y GameServer. Primero se abre el servidor y luego los clientes, si se abre el cliente primero se cerrará automáticamente, si se abren más clientes que los deseados (Ahora mismo 2) el servidor lanzará un mensaje de server is full y echará al último cliente. Si se cierra un cliente y la vacante del cliente queda libre podrá entrar otro jugador. (Si se clickea en la pantalla el círculo del cliente se moverá a la posición del mouse, está posición después del ping se verá corregida debido al servidor sobrescribiendo la información, aunque podemos ver de esta manera qué círculo es el cliente local y qué circulo no. Actualmente no se actualizan las posiciones locales al servidor.**

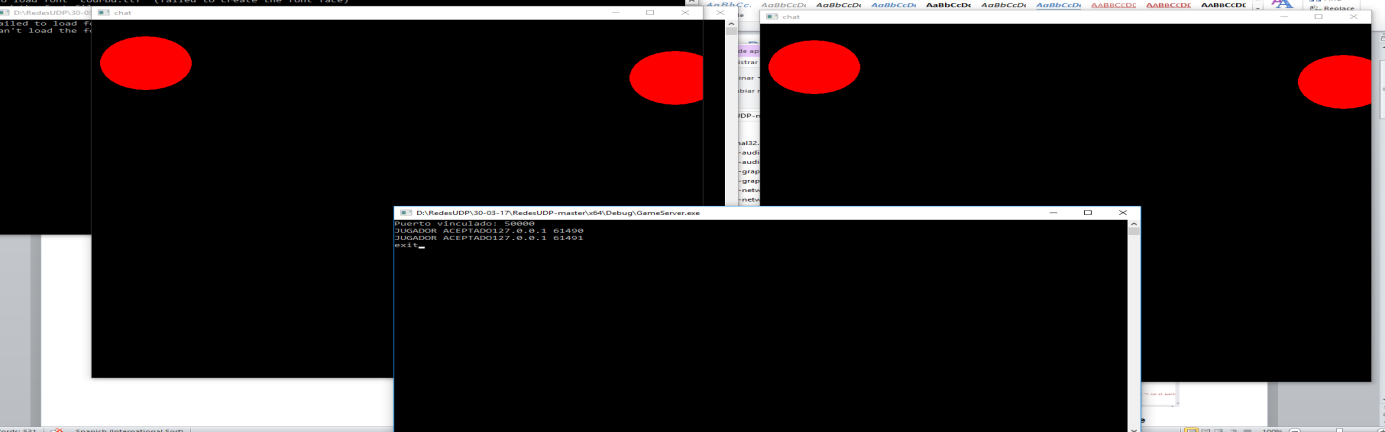
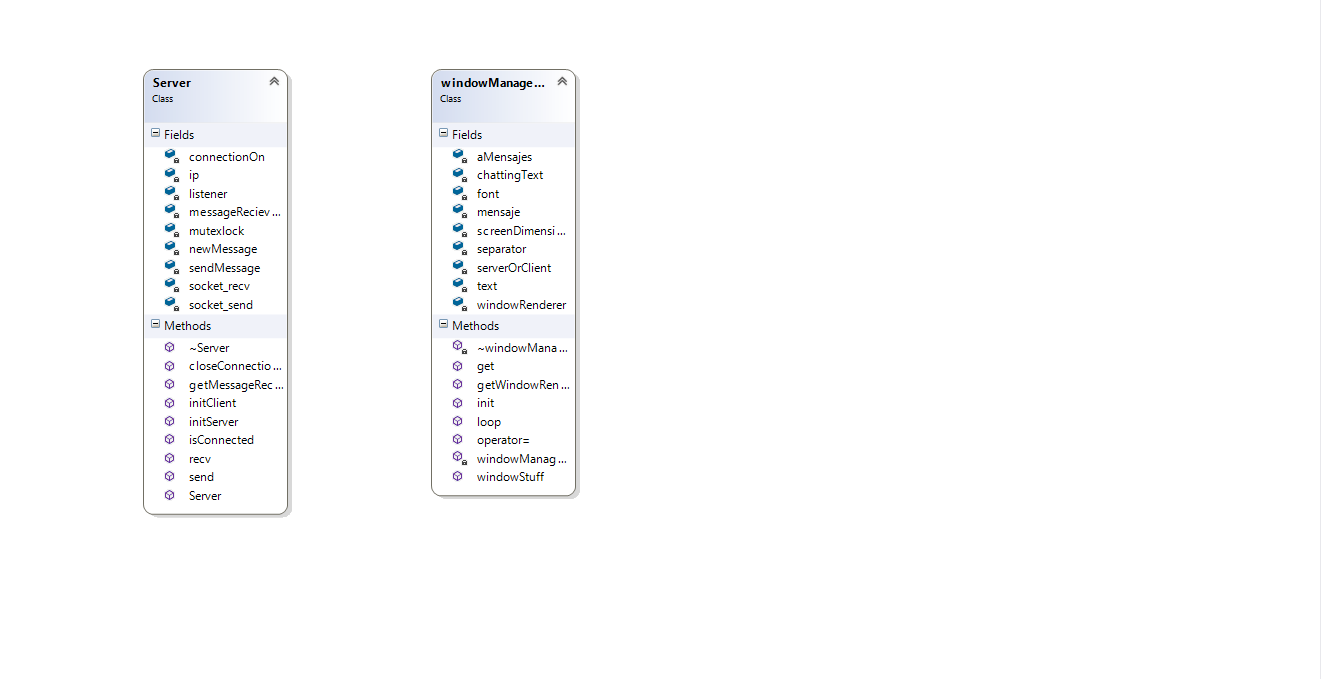
****

Diagrama de clases



**Hay dos proyectos GameClient y GameServer, ambos utilizan la librería GameLib con las clases de InputMemoryBitStream /OutputMemoryBitStream y PlayerInfo. El cliente utiliza otra clase llamada character.h que se encarga de renderizar al jugador y se sincroniza con PlayerInfo en cuanto a las posiciones.**

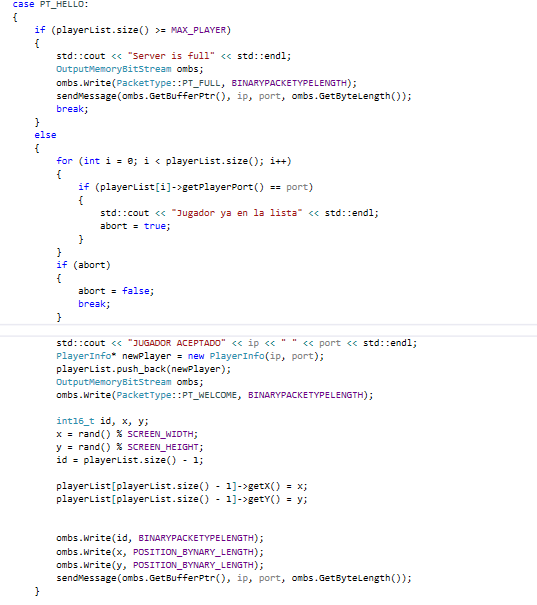
Incluye el protocolo de mensajes utilizado en el establecimiento de conexión. Da las explicaciones extra que creas convenientes.

|  |  |  |
| --- | --- | --- |
| Estructura mensaje | Sentido | Acción Asociada |
| **PT\_HELLO** | **Cliente -> Servidor** | **Pedir al servidor conexión e inclusión en su lista de jugadores.** |
| **PT\_WELCOME +**  **(posición del jugador en vector) + (posición x en el mapa)**  **+ (posición y en el mapa)** | **Servidor->Cliente** | **Aceptar al jugador, darle una posición en el mapa y el conocimiento de qué cliente es.** |
| **PT\_FULL** | **Servidor -> Cliente** | **Desconectar al jugador que quiere entrar cuándo los jugadores superan el MAX\_PLAYER** |
| **PT\_PING** | **Cliente->Servidor** | **Petición del cliente al servidor para actualizar las posiciones de los jugadores en los clientes.** |
| **PT\_PLAYER\_POSITION + id(uint\_16t) + Coordenada X(uint\_16t) + Coordenada Y(uint\_16t)**  **Máximo 10 bits** | **Servidor->Cliente** | **Enviar la posición de algún jugador a cliente.** |
| **PT\_DISCONNECT** | **Cliente->Servidor**  **Servidor->Cliente** | **Desconectar al jugador que haya hecho la petición de PT\_DISCONNECT.** |
| **PT\_SHUTDOWN** | **Servidor->Cliente** | **El servidor desconecta a todos los clientes y posteriormente se cierra.** |

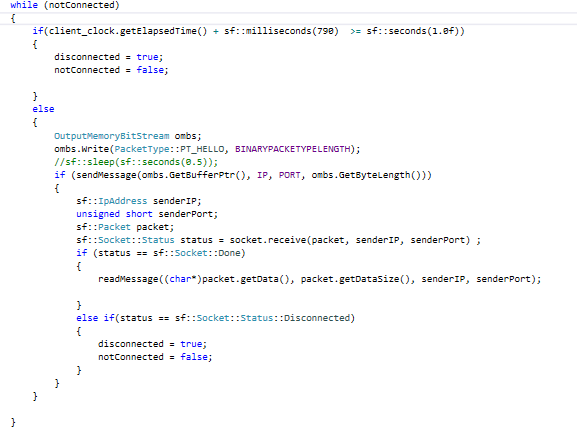
Indica desde qué parte del código estás utilizando cada uno de los mensajes anteriores (clase, función)

**PT\_Hello – Clase : UDPManager(Servidor y Cliente) - initConnection();**

**Servidor**

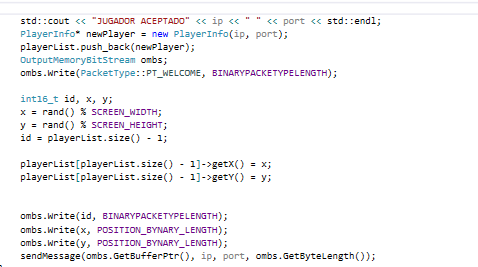
****

**Cliente**

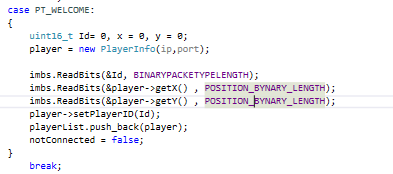
****

**PT\_WELCOME- Clase:UDPManager – readMessage();**

**Servidor**

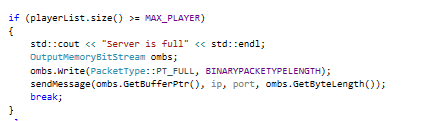
****

**Cliente**

****

**PT\_FULL- Clase:UDPManager – readMessage();**

**Servidor**

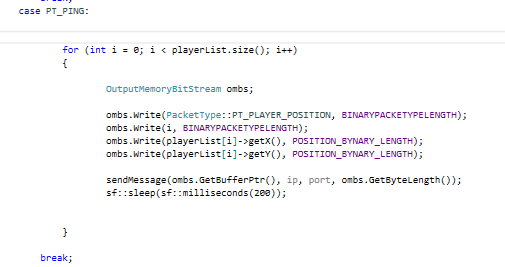
****

**Cliente**

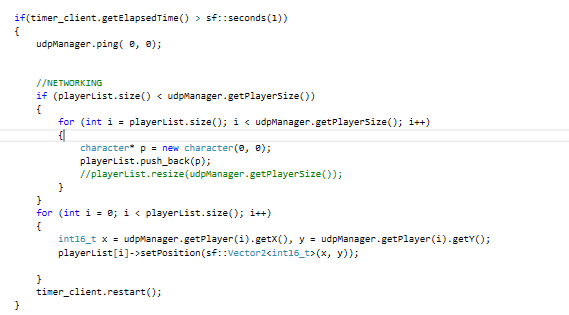
****

**PT\_PING – Clase::UDPManager – ping() – llamado en el windowRenderer (Cliente);**

**Servidor**

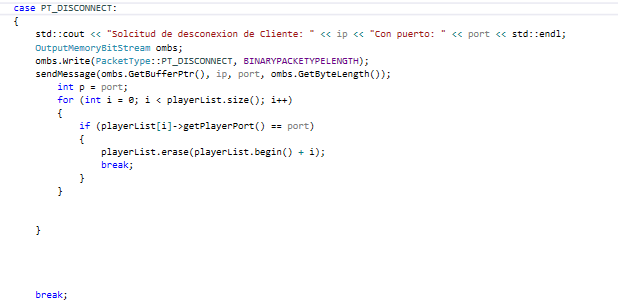
****

**Cliente**

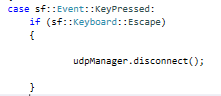
****

**PT\_DISCONNECT – Clase::UDPManager – disconnect() (Llamado desde eventos en windowRenderer) (Cliente) – recv() (server).**

**Servidor**

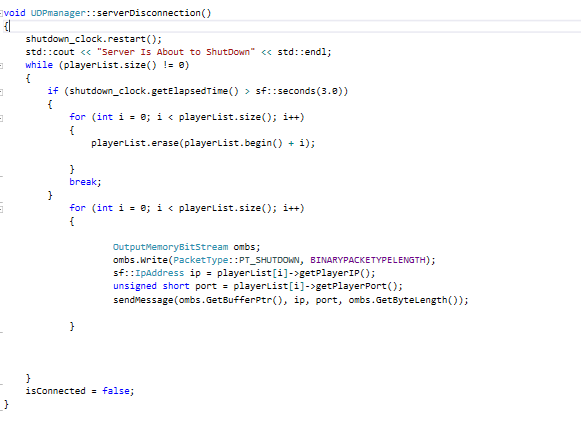
****

**Cliente**

****

**PT\_SHUTDOWN – Clase : UDPManager (Servidor )- serverDisconnection();**

**Servidor**

****

Incluye el protocolo de mensajes utilizado para informar del estado del mapa. Da las explicaciones extra que creas convenientes.

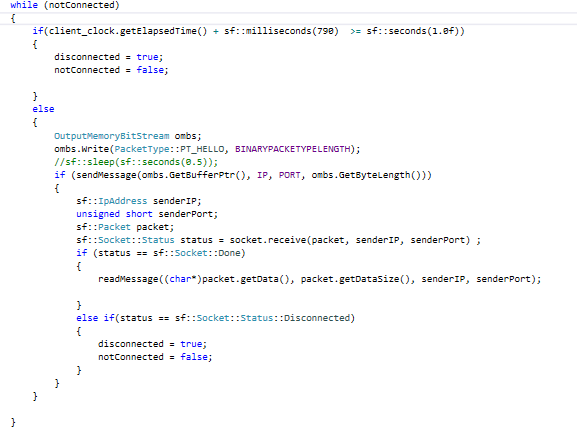
Actualmente sólo se utiliza el **PT\_PLAYER\_POSITION** y el **PT\_PING** para sincronizar las posiciones en el mapa.

Explica cómo y dónde implementas la orientación a conexión para los mensajes del estado del mapa.

**En las funciones de InitConnection de cada UDPManager.**

Indica qué líneas debo comentar para comprobar que el servidor reenvía los mensajes del mapa si el cliente no los recibe.

Todas las funciones de enviar del cliente se envían hasta que recibe validación por parte del servidor. **PT\_HELLO** se envía hasta que reciba el cliente un **PT\_WELCOME**  con un tiempo límite para no crear un bucle infinito ni saturar la red.

****

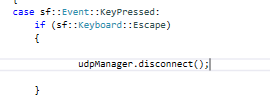
**PT\_PING** al ser una llamada recursiva, si el servidor pierde un paquete volverá a enviarlo debido a la petición continúa de validación del cliente.

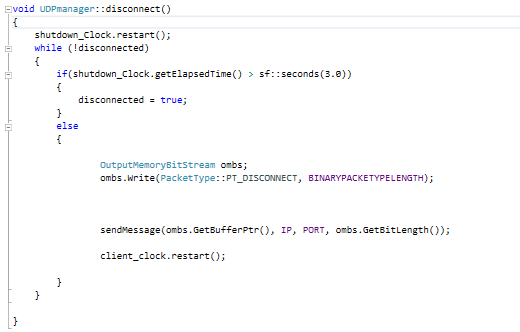
Indica qué técnica sigues para que los clientes se desconecten correctamente.

* Si la desconexión es desde el cliente (clickeando la tecla ESC), el cliente enviará durante unos segundos continuamente señales de desconexión al servidor, cuándo el servidor lo reciba le devolverá una validación para desconectarse correctamente (borrando su personaje del vector del servidor). Si el servidor se ha cerrado indebidamente o ha ocurrido algún error inesperado, el cliente se cerrará después de unos segundos.
* Si la desconexión va desde servidor (Escribiendo en la consola exit o EXIT), el servidor entrará en un bucle de enviar mensajes de desconexión a sus clientes, mientras que siga habiendo clientes seguirá enviando mensajes. Cuándo el size() del vector sea = 0 se cerrará habiendo cerrado todo perfectamente. Si por algún casual pasa algún problema (un cliente se cierra indebidamente o algo por el estilo), el servidor se cerrará después de unos segundos.

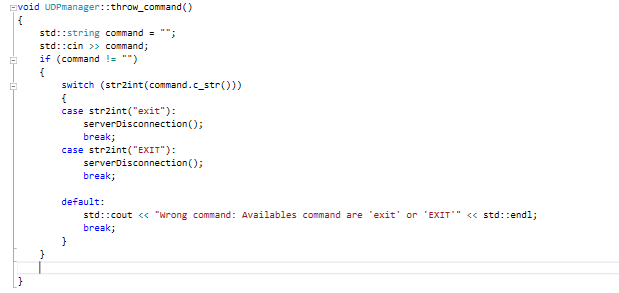
Indica dónde implementas la desconexión de jugadores (en cliente y en servidor)

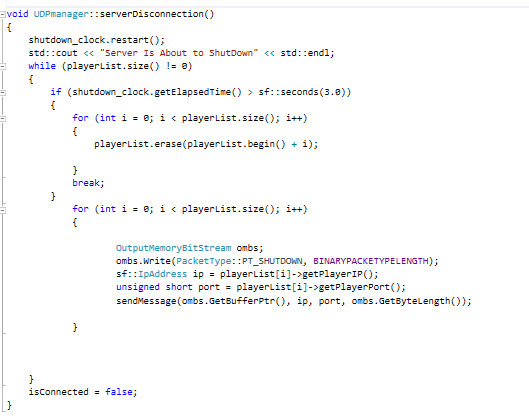
**Si la desconexión es desde cliente.**

****

****

**Si la desconexión es desde servidor.**

****

****

Listado de los mensajes del nuevo protocolo optimizado. Indica cuánto se están optimizando los mensajes en cada caso.

La mayoría de mensajes son uint\_16t por lo que son 16 bits, pero al enviar se optimizan a 10. Además las cabeceras antes se mandaban como std::string y ahora como valores dentro de un enum que en total llegan a 3 bits.

PostMorten

La verdad es que me ha parecido una experiencia positiva, al principio tenía bastante respeto debido a que era UDP y no existía ese colchón de seguridad que nos ofrece TCP, aunque ahora gracias a la optimización y demás puedo ver que en UDP tienes bastante control a la hora de qué envías y cuándo lo envías y la frecuencia a la que lo envías, que he aprendido que es un valor bastante importante y puede suponer que funcione o no el proyecto. Lo que más me ha costado fue la desconexión debido a que pensaba que lo tenía todo calculado, ya que funcionaba y al incluir el ping dejó de funcionar, al estar un rato trabajando en ello me di cuenta que no estaba del todo pulido y había agujeros en los que los paquetes podían caer y no llegar a desconectarse nunca. Probablemente aún queden agujeros en el código pero para ser mi primer programa utilizando el protocolo, creo que no está tan mal. Se podría optimizar más y dejarlo más claro, pero eso lo dejo para trabajo del futuro si me queda tiempo (debido a que este será el código del proyecto).