Taller 6 – UDP (Optimización de datos)

¿Cómo empieza el juego?

**El juego comienza al conectar dos clientes. Uno de ellos aparecerá en la esquina superior derecha y otro en la esquina inferior izquierda. A la misma distancia tiene 5 bolas de las que se tendrá que desprender si quiere ganar la partida, el primer jugador que haga al otro jugador acumular 10 bolas gana.**

¿Cómo evoluciona?

**Es un juego con mecánicas muy sencillas y su evolución depende mucho de los jugadores que lo jueguen. Cuando un jugador aprieta espacio sobre una bola, esta sale disparada en dirección al espacio del contrincante.**

¿Cuáles son las condiciones de finalización y quién es el ganador?

**El primer jugador que acumule las 10 bolas en su lugar del campo pierde.**

¿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.**

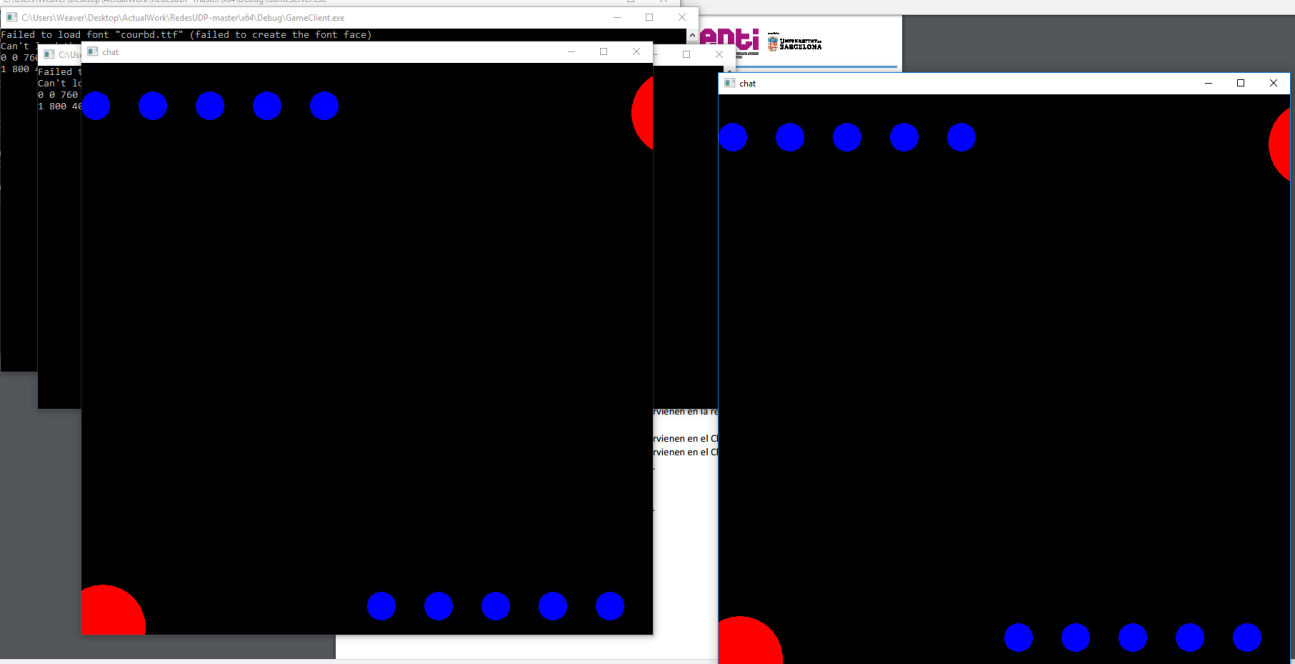
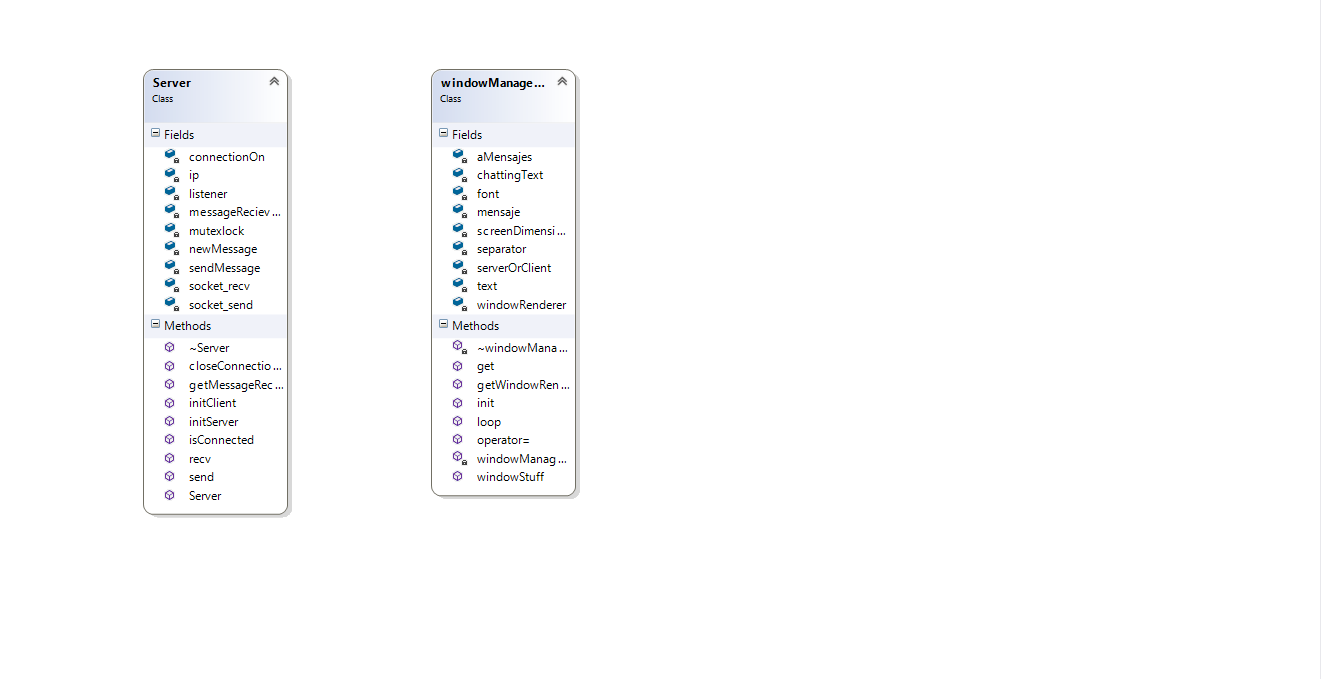
****

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. La clase UDPmanager se encarga de todo lo referente a la conectividad, mientras que la clase WindowsRenderer y character del juego en sí y de renderizar.**

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. Actualmente solo se utiliza al principio.** |
| **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.** |
| **PT\_MOV + id + packid+ calcX(la suma que se ha de hacer a la posición(el acumulado))+completedX (la suma en cliente) + y + negative(bit por si el calcX es negativo) + movementBalls (bolas en movimiento) +(id de bolas en movimiento y dirección).** | **Cliente -> Servidor** | **Envía todo lo necesario al servidor, para que el servidor compruebe que los cálculos hechos por el cliente son correctos.** |
| **PT\_OKMOVE + id + xPos + yPos + failedPosition (booleano que se encarga de ver si el jugador ha hecho un mal cálculo para corregirlo) +(opcional) posición corregida.** | **Servidor->Cliente** | **Verifica la posición del jugador y en caso erróneo la corrige.** |
| **PT\_Ball + número de bolas en movimiento + idBola + dirección.** | **Servidor->Cliente** | **Envía al cliente información acerca de las bolas que están en movimiento para que las mueva.** |
| **PT\_WIN + booleano para saber quién ha ganado (p1 o p2).** | **Servidor->Cliente** | **Envía a los clientes información acerca del ganador, para comenzar el game over.** |

Ítem 3: Define el protocolo de comunicación.

Fase 1: Establecimiento de la conexión y condiciones iniciales

**En esta parte del el servidor escucha completamente y no envía nada hasta que los dos jugadores hayan enviado la información (PT\_HELLO Y PT\_WELCOME). Una vez los dos jugadores han entrado, no envían ningún paquete más y la conexión se queda limpia, en este momento el servidor envía dos paquetes críticos a ambos jugadores (sus posiciones de comienzo). Si el jugador se desconecta en mitad de la partida, el servidor entra en shutdown (Siempre y cuanto se desconecte de manera correcta, con la tecla esc del teclado).**

Fase 2: Desarrollo del juego

**Mensajes críticos:**

**Los mensajes críticos se envían sobretodo bajo la capa del PT\_PING que se envía cada 250 milisegundos. En este mensaje se encuentra la posición de los jugadores y de las bolas que han sido expulsadas.**

**Client Prediction:**

**Al enviar la posición en el PT\_PING se envía un valor acumulado que nos da la clase ClientAccum, la posición del jugador antes de sumar este acumulativo y la posición que tiene actualmente, al llegar al servidor, el servidor calcula su posición de nuevo y si los valores son equitativos, no se envía nada. En caso de que no sean equitativos, el servidor corrige la posición del cliente, mandándole un mensaje de posición errónea.**

**Client Interpolation:**

**Dentro de la clase windowRenderer del cliente se encuentra una función llamada InterpolateMovement(), esta función se encarga de tratar de simular el movimiento del jugador sabiendo su posición final, va sumando la magnitud que el cliente suma a la hora de hacer el input. La posición final se le envía al cliente por medio del PT\_OKMOVE.**

Fase 3: Finalización.

**Intervienen los mensajes con las capsulas de PT\_Disconnect PT\_SHUTDOWN Y PT\_WIN.**

**El cliente va enviando información acerca de las bolas que va lanzando, el servidor va calculando sus posiciones y finalmente, si en uno de estos cálculos alguno de los jugadores ha poseído 10 bolas en su campo, gana el jugador contario. El servidor ya deja de enviar el ping habitual, y envía un nuevo ping en el que sólo envía PT\_WIN cada 250 milisegundos, los clientes entran en el gameOver dónde no pueden moverse ni seguir jugando, se ve al contrincante con todas las bolas y se tiene que apretar escape para cerrarlo todo o cerrar desde servidor con el comando ‘exit’.**

Item 4: Optimización del protocolo de comunicación.

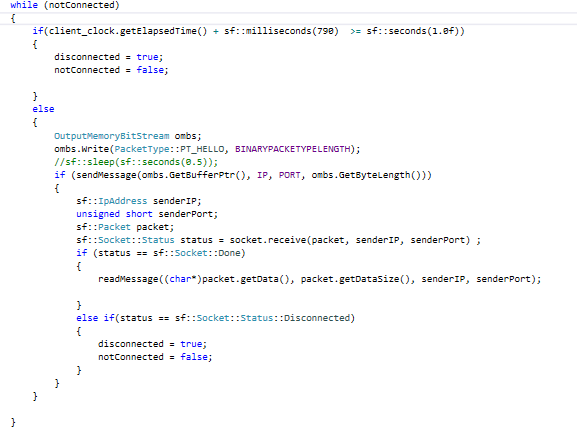
|  |  |  |  |
| --- | --- | --- | --- |
| Estructura mensaje | Tipos | Número de Bits | Explicación |
| **PT\_WELCOME\_id\_x\_y** | **int(enum)\_int\_int\_int** | **4\_4\_10\_10** | **No hay más de 4 bits de enum(PT).**  **No hay ningún array cuya longitud supere los 4 bits por lo que las ids no pueden pasar.**  **Las coordenadas van de 0 a 800. Por lo que con 10 bits va.** |
| **PT\_MOV\_id\_packid**  **\_calcx\_completedx**  **\_y\_negative\_ballssize**  **\_ballid\_balldirect** | **Int(enum)\_int\_int\_int**  **\_int\_int\_bool\_int\_int**  **\_bool** | **4\_4\_10\_10**  **\_10\_10\_1\_4**  **\_4\_1** | **Los calculus de posición de 4 y 10 bits tienen la misma explicación de antes, los que han podido ir con 4 bits van con 4 bits. El packid va con 10 porque los números pueden ascender a esa cifra. Los booleanos son para comprobar si el número es negativo o la dirección de la bola (true arriba) (false abajo).** |
| **PT\_OKMOVE\_ID\_X\_Y**  **\_failedp\_x(opcional)** | **Int(enum)\_int\_int**  **\_bool\_int(op)** | **4\_10\_10\_1**  **\_10(op)** | **Los ints tienen la misma explicación. El booleano es para saber si ha habido un error con la posición del cliente y corregirla, si da false, no se enviará la posición.** |
| **PT\_BALL\_ballsize\_id**  **\_direction** | **Int\_int\_int\_bool** | **4\_4\_4\_1** | **Según el ballsize(la cantidad de bolas que estén activas) se enviarán más ids o menos. El booleano es para saber la dirección.** |

Ítem 5: Client Prediction y t Prediction y t Prediction y Interpolation Interpolation Interpolation

**Para evitar el lag, el juego utiliza la técnica denominada reconciliación en la que el servidor envía la posición (mientras el cliente se está moviendo) y en caso de que el cliente este haciendo algún movimiento indebido lo corrige, pero el cliente tiene la libertad de movimiento sin la corrección permanente del servidor. El movimiento de los otros jugadores es alterado por una interpolación en la que se va sumando la magnitud que suma los clientes al hacer el input del movimiento, esto da una sensación de mayor dinamismo, aunque aun así se nota un poco que está yendo a tirones.**

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.

****

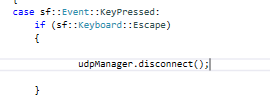
**PT\_PING** al ser una llamada recursiva, si el servidor pierde un paquete volverá a enviarlo debido a la petición continúa de validación del cliente.

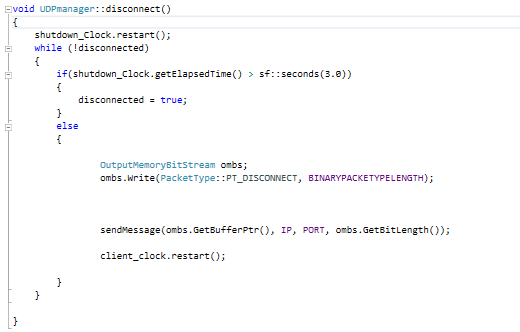
Indica qué técnica sigues para que los clientes se desconecten correctamente.

* 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.

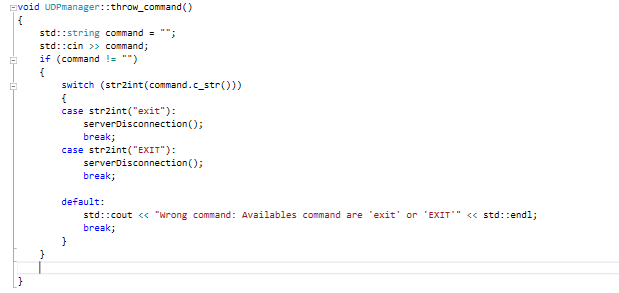
Indica dónde implementas la desconexión de jugadores (en cliente y en servidor)

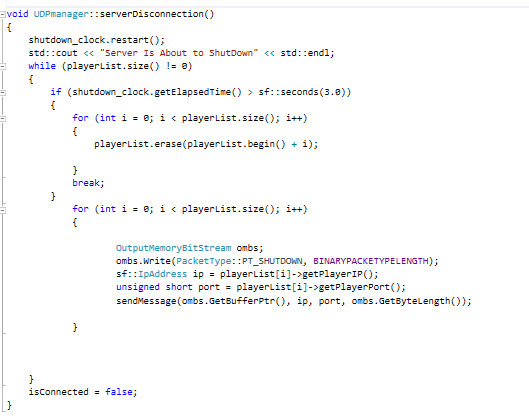
**Si la desconexión es desde cliente.**

****

****

**Si la desconexión es desde servidor.**

****

****

Post-Mortem

Hemos encontrado dificultades a la hora de implementar las bolas sobre todo por una mala organización de tiempo y por habernos puesto a ello un poco tarde, disfrute bastante implementando el protocolo de interpolación, de predicción y de renconciliación. Aun así me llevo una grata experiencia, debido a que he aprendido más acerca de la naturaleza de UDP y un poco más de cómo enviar y cómo no mensajes y una grata experiencia debido a que me ha devuelto las ganas de volver a programar como solía hacerlo (aunque este código este hecho un poco a la prisa). El trabajo no tiene el acabado que me gustaría, pero al menos está acabado.