ESTUDIO RORSCHACH

Diseño técnico del funcionamiento del motor de red
Last Bear Standing

Miguel Paniagua Muela
Miguel Córdoba Alonso
José María Ortiz García
José Roberto Martínez Gras
Jorge Puerto Esteban
Manuel Gómez Cámara

Índice

Diseño de la Arquitectura

La arquitectura de red sigue un modelo cliente/servidor. Tanto el cliente como el servidor actúan independientemente del otro y se comunican con mensajes. Es una arquitectura diseñada para el funcionamiento en LAN. Con un máximo de 4 jugadores por partida.
La red multijugador emplea Raknet, un motor de red en C++, para facilitar la comunicación entre el cliente y el servidor, ya que Raknet proporciona los protocolos UDP y TCP.

Diseño y lectura de paquetes

El **paquete** de información está compuesto por un **sistema de dirección**, que permite obtener información imprescindible del paquete como IP y puerto, un **GUID** que lo identifica, **tamaño** del paquete en bits y bytes y por último los **datos enviados** como char.

Como se explica en el documento de Diseño de requerimientos y funciones de red, los paquetes se envían por eventos de teclado, el **puerto** lo asigna el cliente y el **tamaño** es el adecuado al **mensaje** por lo tanto, vamos a centrarnos en la lectura de éste último.

El mensaje sigue una estructura definida. La información se establece en un orden determinado y separada por espacios. La primera parte del mensaje siempre es la misma, corresponde al tipo de mensaje (esto se explicará más adelante), después la ID del jugador que envía el mensaje y el resto de información dependerá del tipo de mensaje.

El cliente recoge el paquete y procede a interpretar su información. Para ello corta el mensaje a partir de los espacios, y llama a la función correspondiente en función del tipo de mensaje que ha interpretado.

EJEMPLO DE MENSAJE:

**En este caso cliente identifica que debe llamar a la función correspondiente a recibir un salto.*

Diseño técnico

Servidor

El servidor recibe los mensajes y actúa en función del mensaje del paquete:

1. **ID_NEW_INCOMING_CONNECTION:**
   En este caso el servidor envía los clientes la ID del usuario recién conectado y la de los jugadores conectados actualmente.
2. **ID_CONNECTION_LOST:**
   El servidor notifica de la perdida de conexión y envía la dirección del cliente desconectado, al resto de clientes.
3. **ID_DISCONNECTION_NOTIFICATION:**
   Notifica de la desconexión normal de un cliente.
4. **ID_INCOMPATIBLE_PROTOCOL_VERSION:**
   Notifica la incompatibilidad de versiones en la conexión.
5. **OUT_OF_PROTOCOL:**
   El servidor envía en broadcast el mensaje enviado por el cliente.

Este último paquete será el más frecuente y el encargado de mantener el juego actualizado en cada cliente.

Cliente

El cliente se encarga tanto de la parte de **leer e interpretar** un paquete como de la de **confeccionarlo y enviarlo**.

En cuanto a la parte de **leer e interpretar** un paquete el cliente recibe la información del paquete como se especificó anteriormente. Una vez leído y almacenado lanza la **función correspondiente** al **tipo de paquete** recibido:

1. **Analizar_Paquete_0**
   Función correspondiente a la conexión de un nuevo jugador. El cliente almacena los jugadores en red y sus respectivas ID.
2. **Analizar_Paquete_1**
   Función correspondiente la actualización cada 1-5 segundo/s* de un jugador por una posible pérdida de sincronización en el tiempo. El cliente se encarga de actualizar cada una de las variables de estado del *player* que simula al cliente que envía el paquete.

*El envío de este paquete puede variar de 1 a 5 segundos por periodo de test.

3. **Analizar_Paquete_2**
   Función correspondiente a la llamada al método usarArma del *player* que simula al cliente que envía el paquete.
4. **Analizar_Paquete_3**
   Función correspondiente a la llamada al método CogerTirar un arma determinada del *player* que simula al cliente que envía el paquete.
5. **Analizar_Paquete_4**
   Función correspondiente a la llamada al método Mover del *player* que simula al cliente que envía el paquete.
6. **Analizar_Paquete_5**
   Función correspondiente a la llamada al método Saltar del *player* que simula al cliente que envía el paquete.
7. **Analizar_Paquete_6**
   Función que lanza el inicio de partida.
8. **Analizar_Paquete_7**
   Función correspondiente a la llamada al método Morir del *player* que simula al cliente que envía el paquete.
9. **Analizar_Paquete_8**
   Función correspondiente a la llamada al método FingirMorir del *player* que simula al cliente que envía el paquete.

En cuanto a la parte de **confeccionar y enviar** un paquete el cliente decide qué tipo de paquete crear en función del evento recibido en *player*:

1. **Enviar_General**
   Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_1.**
2. **Enviar_Usar**
   Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_2.**

3. **Enviar_Salto**
   Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_5.**

4. **Enviar_Cogido**
   Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_3.**

5. Enviar_Moviendo

Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_4.**

6. Enviar_Muerto

Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_7.**

7. Enviar_HacerseMuerto

Función correspondiente al envío al servidor de un paquete con información necesaria para que los clientes que lo reciban llamen a su **Analizar_Paquete_8.**