Proyecto Final. Redes y Videojuegos en Red

Instrucciones para la entrega

- **Rellenar esta memoria** con la memoria del proyecto.
- Si el proyecto se realiza en **pareja** sólo es necesario **rellenar una memoria del proyecto**.
- El **código** del proyecto se entregará usando www.github.com.
- La **evaluación** del proyecto tendrá en cuenta el último commit antes de la fecha de entrega. No se considerarán commits posteriores.
- El **proyecto deberá defenderse en una presentación con el profesor**. En el acto de presentación los alumnos responderán preguntas sobre el proyecto realizado.

Formulario de entrega

Título	Space Wars	
Alumno	Apellidos y Nombre	Felipe Cuadra Plaza
	Usuario github	Alonefcp
Alumno	Apellidos y Nombre	Antonio Jesús Guerra Garduño
	Usuario github	Antgue01
URL del repositorio		https://github.com/Antgue01/Space-Wars

Memoria del Proyecto

Breve Descripción

Space Wars consiste en una batalla entre dos naves espaciales controladas por dos jugadores, las cuales tienen cada una, tres vidas y un escudo.Si la vida de uno de los jugadores llega a cero se acaba el juego.

Los jugadores pueden efectuar tres tipos de disparo: el normal, que simplemente daña a lo primero con lo que colisione, el circular, que consiste en disparar cuatro disparos alrededor del jugador que se van alejando y un láser que rebota con lo primero con lo que colisiona.

El escudo puede recibir dos golpes antes de destruirse y cuando esto sucede se recarga con el tiempo (2 segundos), tiempo durante el cual no se puede utilizar.

El escenario está poblado con asteroides que hay que evitar, pues hacen daño si colisionan con el jugador. Si un asteroide colisiona con una bala este se divide en dos asteroides más pequeños, hasta tres veces.

La vida de los jugadores aparece en pantalla y cuando uno de los dos gana aparece un texto con el nombre del ganador (servidor o cliente).

1. Arquitectura del Juego

1.1 Modelo del juego

El juego sigue la topología cliente-servidor en la que uno de los jugadores es el servidor y se decide cuál de los jugadores es el servidor mediante el último argumento del programa, el cual si es un 0 indica que es el servidor y si es un 1 es el cliente. El juego no comienza hasta que no están conectados ambos porque el servidor se queda esperando hasta que se conecte un cliente.

1.2 Estado del juego, objetos y replicación

Utilizamos dumb terminal[3]. El input se guarda en un vector de booleanos los cuales cada uno de ellos indica una acción. Este input en el caso del cliente se mandará al servidor para ser procesado por este.
El servidor, por su parte, también procesa su propio input que se recoge en un vector de booleanos. Actualiza posteriormente el estado del juego y manda una copia de este al cliente.

Para la recepción y envío de los mensajes tenemos dos clases: MessageQueue (tiene dos colas,una para mensajes a enviar y otra para mensajes a recibir) y NetManager. MessageQueue recibe por constructora los dos sockets implicados en la comunicación, en caso del cliente son los dos el mismo socket y en el servidor uno es el suyo y el otro es el del cliente.

El bucle principal del juego es el siguiente:

-Recepción de mensajes: lo primero que se recibe es un tipo de mensaje llamado CountMessage que contiene el número de mensajes que van a llegar. El NetManager le dice a la cola que los reciba todos y esta rellena su cola de mensajes a recibir. Los mensajes se reciben por pares ,el primero de tipo TypeMessage que indica qué entidad hay que instanciar y el otro mensaje es el propio estado de la entidad. En caso de que unos de los mensajes sea el de terminar el juego se comunicaría a la

clase principal del juego mediante el valor de retorno de la función recieve de la cola y del NetManager. Una vez que ha terminado de recibir mensajes, el NetManager reparte los mensajes recibidos a todas las entidades.

-Recoger input: si recibe el input para salir del juego lo pone en la cola de mensajes para enviar y si es otro tipo de input cada entidad lo recoge por separado.

-Actualizar: cada entidad sabe si es parte del cliente o del servidor y se actualiza o no dependiendo de esa información. Solo si eres servidor se comprueban las colisiones mediante la clase GameLogic, que no es una entidad.

-Pintar: todas las entidades se pintan de la misma forma, tanto en el servidor como en el cliente.

-Enviar: lo primero que se recibe es un CountMessage que contiene el número de mensajes que van a enviar. El NetManager recorre todas la entidades y para cada una pone en la cola, un mensaje con su tipo (de tipo TypeMessage) y otro con su estado. Una vez terminado se envían todos los mensajes por red, vaciando la cola en el proceso.

1.3 Protocolo de aplicación y serialización

Tenemos una clase Serializable con los métodos to_bin y from _bin que deben ser rellenados para llevar a cabo la serialización. Además tenemos otra clase abstracta llamada Entity que hereda de Serializable y le añade un id para identificar la entidad, un enum para saber el tipo de mensaje que envía o recibe y un bool inUse para saber si la entidad está o no activada.

Entity también añade a Serializable los métodos abstractos update, que sirve para actualizar el estado de la entidad, draw, que sirve para pintar la entidad y deliverMsg, que sirve para guardar los atributos que recibe por red.

Todas nuestros objetos del juego heredan de esta clase y tienen que rellenar los métodos abstractos para que todo funcione. Aparte tenemos tres tipos de mensajes que no son entidades pero heredan de Serializable que también se envían y reciben. Estos son CountMessage, que tiene el número de mensaje a enviar o recibir, TypeMessage, que contiene el tipo de la entidad y LoginMessage, que el cliente envía cuando se conecta al servidor.

2. Diseño del Servidor

Al inicio se queda esperando hasta que se conecta un cliente; es decir hasta que llegue un mensaje de tipo LoginMessage. Entonces se guarda el socket del cliente y le envía una copia de su estado. Esto se hace para iniciar el bucle del juego, ya que, en ambos casos empieza por recibir.

El servidor es el encargado de ejecutar la lógica del juego, y cada entidad sabe si es parte del servidor o del cliente. Si forman parte del servidor descarta todo lo que llega menos el input. Además la clase GameLogic, que controla las colisiones, se instancia solo en el servidor.

Nuestra implementación no usa hilos.

Cada frame tiene que durar 10 ms como mínimo y si no se espera hasta llegar a los 10 ms

3. Diseño del Cliente

Lo primero que hace el cliente es enviar un mensaje de tipo LoginMessage para identificarse. No ejecuta

ningún tipo de lógica, ya que recibe todo del servidor. Simplemente se dedica a renderizar el estado del juego y recoge su propio input para mandarselo al servidor. Cuando recibe un mensaje descarta el input y recoge la lógica. La implementación de este tampoco usa hilos.

4. Conclusiones

En cuanto a los aspectos más importantes del proyecto, la claseNetManager, que controla que cada entidad reciba el mensaje que le corresponde, junto con el método deliverMsg, que gestiona el contenido de los mensajes que se envían y se reciben, facilitan la implementación de la parte de red del juego.

Además MessageQueue abstrae el paso de mensajes, ya que simplemente recibe los sockets con los que tiene que enviar y recibir. El cliente pasará unos y el servidor otros.

Respecto a las mejoras, usaríamos hilos, uno para el input y otro para la lógica del juego y red, tanto en el servidor como en el cliente. También usaríamos más la herencia, ya que haría el código más legible. Además parametrizaríamos más ciertas clases del juego como la clase Vessel, en la que las únicas teclas que se toman como parámetro son las del movimiento y podría tener el resto del input parametrizado. Asimismo podríamos hacer la arquitectura del juego por componentes, ya que al principio lo intentamos pero no fuimos capaces de hacerlo. También podríamos incluir sonidos y manejar cómo hacer que se mande por red el momento en el que tiene que reproducirse un sonido determinado.

Referencias

1.- Hemos usado como base del proyecto otro proyecto de la asignatura llamada Tecnología de la programación de videojuegos 2, del cual hemos reutilizado las clases que gestionan la librería SDL, el renderizado de los objetos, el movimiento de las naves, asteroides y balas, la división de los asteroides, parte de la detección de colisiones, concretamente la colisión bala-asteroide y asteroide-nave y el uso de pools para balas y asteroides.

2.- Página donde vimos como instalar SDL, SDL image, SDL mixer y SDL TTF en linux: https://gist.github.com/BoredBored/3187339a99f7786c25075d4d9c80fad5

3.- También usamos la clase Socket y Serializable de la última práctica y las diapositivas del tema 3.1.