Taller de Programación I (75.42)

Primer cuatrimestre de 2021

Trabajo Práctico Final
Counter Strike 2D

Documentación técnica

Alumno	Padrón	Email
Ernesto Alvarez	102221	eralvarez@fi.uba.ar
Matías Huenul	102135	mhuenul@fi.uba.ar

Índice

1. Introducción

2. Cliente

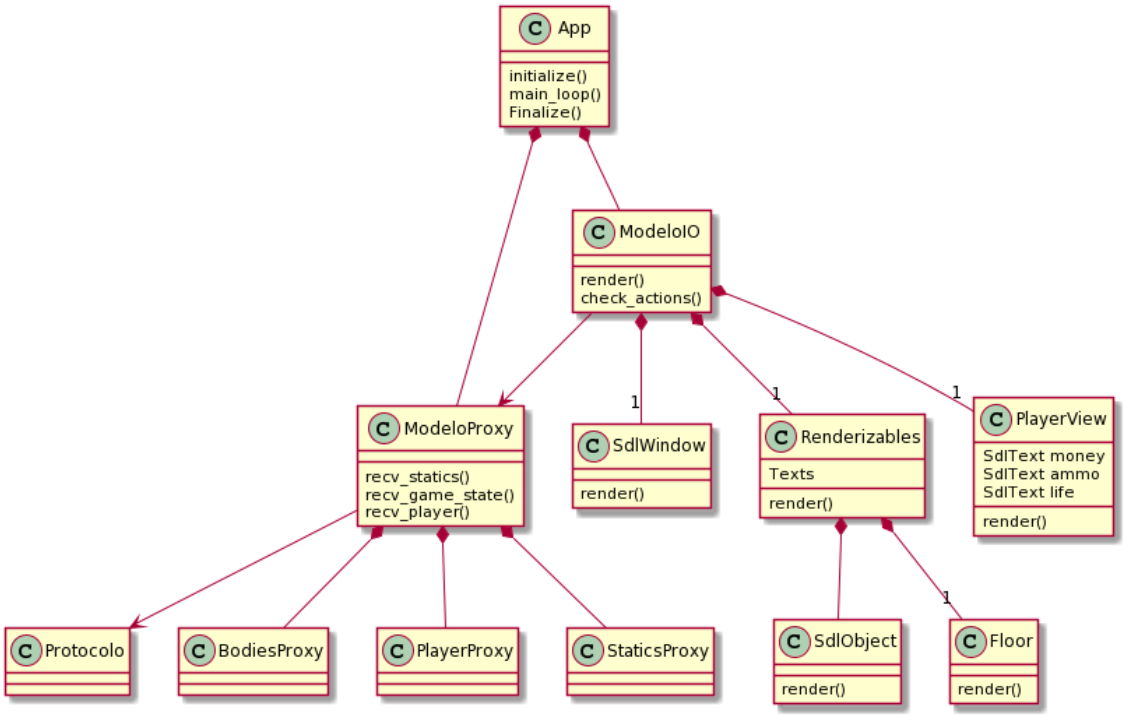
La aplicación que el usuario utilizará esta compuesta por 2 modelos principales: El ModeloIO y el modeloProxy.

El ModeloIO es el encargado de todo lo que el usuario podrá experimentar: Vista, controles y sonidos. Este se comporta como en otro tipo de servicios se denominaría "front-end".

El ModeloProxy, por su parte, es el encargado de comunicarse, a traves del socket, con el servidor. Decodifica la información que llega constantemente y la almacena para que el ModeloIO pueda utilizarla. También el envio de acciones al servidor, que el usuario genera, es obligación de este.

Por otro lado, estructuralmente, tenemos 3 hilos:

- El hilo principal, en el que se ejecuta la parte no bloqueante del programa: Renderización, arbol de decisiones del programa en base a la información que el servidor le proveyó, etc.
- El hilo *sender*, en el que se recibe el input del usuario, y se envia posteriormente al servidor. Utilizando funciones del ModeloProxy para ello.
- El hilo *receiver*, en el que se recibe la información del servidor, que luego se almacenará en el ModeloProxy.



2.1. Vista y controles del programa

Para la renderización en la pantalla, la reproducción de sonido y la captación de acciones por parte de los perifericos, hemos decidido utilizar el framework Simple Directmedia Layer (SDL) en su version 2.0. Ya que englobaba la mayor parte de nuestras necesidades para continuar con el videojuego.

Para el sonido se ha utilizado la libreria sdl_mixer, para los textos que se observan por pantalla la libreria sdl_ttf, para la utilización de imagenes como método de modelado la libreria sdl_image.

Este framework, si bien provee las funcionalidades que se necesitaban, no es basado en clases y por lo tanto tampoco con clases RAII. Por lo que se decidió envolver estos componentes que el framework brindaba en clases orientadas a objetos que brindaran un funcionamiento RAII.

2.2. Renderizado

Como parte del renderizado, se hace una selección de las texturas que el cliente tendrá que renderizar, creando una sola textura por cada tipo de objeto que se renderizará por pantalla. Esta textura será reutilizada a lo largo de la ejecución multiples veces, siempre que sea necesario. Por lo que cada tipo de objeto dentro del juego, solo tendrá una textura asociada por parte de la vista, lo unico que se modifica es: Cuantas veces será necesario renderizarla por pantalla y en qué posiciones.

2.3. Representación

Los objetos que dentro del juego se observan, se decidió dividirlos en 3 partes:

- Los objetos estáticos: Son los objetos que el mismo mapa ya tendrá, por ejemplo cajas o paredes, y estos solo serán recibidos una vez del lado del servidor y se almacenarán en el ModeloProxy. Estos objetos, sus posiciones y texturas, no serán modificados a lo largo del juego, su información será siempre la misma.
- Los objetos dinámicos: Son los objetos que sí tendrán cambios a lo largo de la partida, estos serán actualizados conforme a lo que el servidor ordene. Sus posiciones, angulos o texturas pueden ser alterados y por ello el servidor enviará su información constantemente para poder conseguir el movimiento deseado. Ejemplos: Otros jugadores, armas o la bomba.
- El jugador: Si bien es similar a un objeto dinámico, la cantidad de datos que este necesita es mayor a la cantidad de un objeto comun y corriente. Por lo que el jugador/usuario es tratado como una entidad aparte. Parte de está información es su vida, el arma que posee, la cantidad de balas, etc.

3. Servidor

El servidor está conformado por dos hilos principales que se comunican a traves de una cola no bloqueante. Cuando un cliente se conecta, uno de estos hilos, Lo-

ginHandler, lo acepta y agrega a la cola un objeto Login, que contiene la informacion enviada por el cliente (la partida a la cual desea unirse y el equipo elegido).

El otro hilo, Acceptor, se encarga de obtener elementos de esta cola, y al encontrar un nuevo Login, primero valida sus datos (básicamente, que el equipo al que quiera unirse tenga aún espacio y que la partida no haya comenzado) y luego lo registra como un nuevo Peer. Luego, en cada iteración, avanza el estado de todas las partidas existentes, notifica a cada cliente de estos estados y ejecuta los comandos recibidos.

Ahora bien, el envío de los estados y la recepción de comandos no se realizan en el hilo Acceptor. Cada Peer contiene dos hilos: un Sender y un Receiver.

Cuando desde el Acceptor se obtiene el nuevo estado de la partida, se lo agrega a la cola de estados. Luego, el Sender lo obtiene de esa cola y lo envía al cliente correspondiente. Cabe destacar que la cola de estados es bloqueante: el Sender no tiene nada que enviar hasta recibir un nuevo estado.

Por otro lado, el Receiver de cada Peer se encarga de recibir constantemente comandos de su cliente para agregarlos a la cola de comandos. Entonces el Acceptor, al obtener comandos de esta cola los aplica en la partida. En este caso, la cola es no bloqueante: el Acceptor debe poder seguir trabajando, hayan o no comandos nuevos, para que el tiempo pueda seguir transcurriendo en la partida y para simular la física del mundo.

3.1. Lógica de las partidas

Cada partida esta representada por una instancia de la clase Game, la cual contiene toda la información acerca de la misma, los jugadores y el mundo. Esta clase dispone de métodos para consultar y actualizar su estado, y para ejecutar comandos sobre un jugador dado.

3.2. Física del juego

La física del juego está controlada por completo por el framework Box2D, el cual se encarga de simular los movimientos de los distintos tipos de cuerpos y detectar las colisiones. En particular, se implementaron las clases World y Body como wrappers sobre las clases dadas por Box2D.

World representa al mundo donde transcurre la partida. Un World posee cuerpos y se encarga de actualizarlos en cada iteracion (step).

Body es una clase abstracta y representa a un cuerpo genérico presente en un mundo. Un Body tiene una posición, un ángulo, y puede ser estático o dinámico, y en este último caso tendrá una velocidad. Además exige a sus clases derivadas que implementen el método handleCollision para cada tipo específico de cuerpo. De esta forma, cuando Box2D detecta una colisión (a través de ContactListener), estará definido el comportamiento entre cualquier par de cuerpos: por ejemplo, cuando una bala (Bullet) impacta a un jugador (Player), éste último perderá vida (en caso de que el disparo haya provenido de un enemigo) y el jugador que disparó perderá munición.

4. Protocolo de comunicación

Para la comunicación entre servidor y clientes, se diseñó un protocolo binario que permite intercambiar la información necesaria en cada etapa del juego.

4.1. Login

Lo primero que el cliente envía al conectarse al servidor es la información del login: la partida y el equipo al que desea unirse.

4.2. Configuración

En caso de login exitoso, lo siguiente a comunicar es la configuración de la partida, la cual está compuesta por el mapa y los parámetros del stencil (ángulo y radio). Esta información sólo se envía una vez para ser almacenada y usada por el cliente en toda la partida.

4.3. Estado del juego

Luego de la configuración, se continua con el estado del juego, que está compuesto por la información del jugador controlado por el cliente (posición, ángulo, vida, munición, arma equipada) y de los cuerpos presentes en el mundo (posición, ángulo y tipo). Además, cuando una ronda finaliza, se envía el id del equipo ganador. El estado se envía en cada iteración del juego, mientras la partida continue.

4.4. Finalización de la partida

Cuando la última ronda finaliza, y con ella la partida, se envía el estado final, el cual está formado por las puntuaciones de todos los jugadores (dinero ganado, asesinatos y muertes) y el id del equipo ganador.

5. Licencias

5.1. Logger

Como ayuda para la detección de bugs asi como para poder inspeccionar la ejecución del programa más detalladamente, se utilizó un sistema de loggeo hecho por un tercero.

Esta libreria es pertenencia de **Pankaj Choudhary** bajo la licencia CPOL. Para más, ingresar aquí. Dejamos en claro que esta libreria no nos pertenece y su código fue utlizado para fines no ilegales.