Argentum Online
Documentacion Tecnica

Taller de programacion I [75.42]
Primer cuatrimestre de 2020

Taiel Colavecchia - 102510 - tcolavecchia@fi.uba.ar
Franco Daniel Schischlo - 100615 - fschischlo@fi.uba.ar
Nicolás Aguerre - 102145 - naguerre@fi.uba.ar

Índice

1. Requerimientos de Software

1.1. Sistema operativo

Para compilar y ejecutar el proyecto, se debe utilizar alguna distribución de GNU/Linux de 64 bits. Se recomienda utilizar en particular alguna distribución basada en Debian.

1.2. Herramientas basicas

Tanto para compilar el cliente como el servidor, es necesario instalar GCC, Make (ambas incluidas en el paquete build-essential) y CMake. Cada una de estas puede instalarse mediante:

```
$ sudo apt-get install build-essential
$ sudo snap install cmake --classic
```

1.3. Bibliotecas

1.3.1. Cliente

Para compilar/desarrollar el cliente, se deben instalar las versiones de desarrollo de SDL. Esto puede ser logrado mediante

```
$ sudo apt-get update
$ sudo apt-get install libsdl2-dev libsdl2-image-dev libsdl2-ttf-dev libsdl2-mixer-dev
```

1.4. Servidor

Para compilar/desarrollar el servidor, se debe instalar la biblioteca *boost*.

```
$ sudo apt-get install libboost-all-dev
```

1.5. Para la creacion y edicion de mapas

Para crear o editar mapas, es necesario instalar el software Tiled.

```
$ sudo snap install tiled
```

2. Descripción general

El proyecto es en esencia una aplicación distribuida, compuesta por dos programas: Un cliente y un servidor. A rasgos generales, el cliente no es mas que una "ventana"que muestra el estado del juego en cada momento, donde dicho estado se encuentra en realidad en el servidor, donde también se actualiza.

Como se trata de un juego multijugador, el servidor es capaz de atender varios clientes de forma concurrente, mientras que la conexión desde el cliente hacia el servidor es única (es decir, el cliente solo realiza *una* conexión con el servidor).

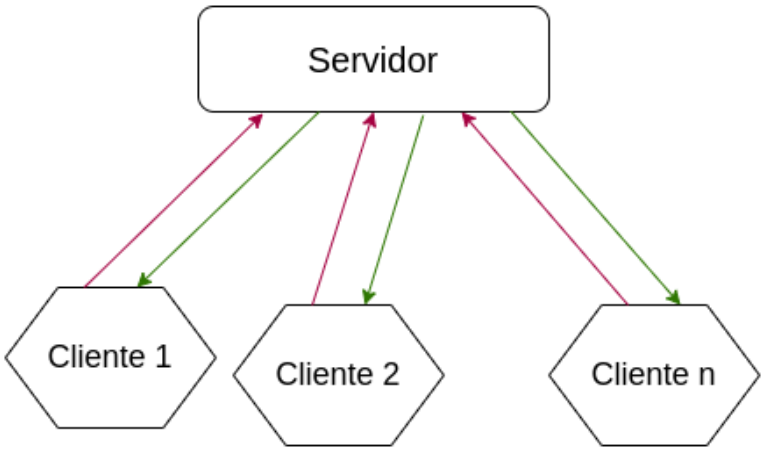

Figura 1: Diagrama general de la aplicación.

Como se mencionó, el cliente no es mas que una ventana al estado del juego que en realidad se encuentra en el servidor. Ninguna de la lógica del juego es calculada en el cliente. Los eventos del usuario (como por ejemplo aquellos que se desencadenan al presionar una tecla) son encapsulados por el cliente en una forma conocida por el servidor, y enviados a través de la red, para que sea el servidor quién decida como debería cambiar el estado del juego de acuerdo a la acción realizada por dicho jugador.

La comunicación entre el servidor y los clientes está dada por mensajes de eventos que se describirá más detalladamente en la sección de Eventos.

Desde el punto de vista del cliente, no siempre es posible enviarle al servidor las mismas acciones. Por ejemplo, a un jugador que aún no se conectó al juego con el nombre correspondiente a su personaje no le es posible indicarle al servidor que se desea mover a la izquierda (porque, después de todo, no tiene un personaje asignado). Esto nos lleva al concepto de *vistas* en el cliente, que se corresponden con las diferentes configuraciones de "ventanas"que ve el jugador mientras tiene abierto el cliente. Existen tres vistas diferentes en el cliente: La vista de inicio de sesión, que es capaz de enviar el mensaje correspondiente a la conexión inicial con un nombre de personaje; La vista de creación de personaje, capaz de enviar una solicitud de creación de personaje con los atributos deseados; y por último, la vista principal de juego, a partir de la cual se pueden enviar mensajes de movimiento, ataque, chat, etc. Cada una de estas vistas, así como también los mensajes, serán explicados con mas detalle en la próxima sección.

El servidor consta principalmente de tres partes, una encargada de mantener *actualizado* el estado del juego (la llamaremos *"GameLoop"*), una segunda encargada de atender las acciones que quieren realizar los clientes sobre el estado del juego (*"Dispatcher"de eventos*) y una última que envía las actualizaciones del juego a los clientes correspondientes (*Öbserver"del mapa*). De las

primeras dos el servidor cuenta con una única por para todo el servidor, mientras la de la última hay una única para cada mapa del juego.

3. Cliente

3.1. Primitivas de SDL

Todas las gráficas del cliente están construidas en base a un conjunto de primitivas de SDL, que no son mas que wrappers de las estructuras de SDL hechas para C, pero adaptadas para ser clases RAII de C++. Una referencia detallada de estas clases (y de todas las del cliente) puede ser hallada en la referencia del cliente.

<Diagrama de clase>

3.2. Engine

Una capa de abstracción por encima de las primitivas de SDL, se encuentra el *engine*. El *engine* construye, a partir de estas primitivas, un conjunto de clases que representan objetos con distintas características dentro del juego, como ser los conjuntos de sprites que representan el movimiento de un personaje en todas las direcciones, las imágenes estáticas que componen el mapa, los elementos de la interfaz de usuario, etc. Todas las unidades en el engine están expresadas en términos de las unidades arbitrarias del juego (tiles). Cada uno de estos elementos será explicado con mas detalle en esta seccion.

3.2.1. Camara y objetos Renderizables

El engine establece que todo objeto renderizable dentro de la vista principal del juego debe heredar de la clase RenderizableObject.

<Diagrama de clase>

Los dos objetos renderizables estandar son los Actors y las Decorations
Los Actors se utilizan para renderizar aquellos sprites que representen las cuatro posibles orientaciones de un mismo objeto (y se utiliza para los personajes).

Las Decorations se utilizan para renderizar cosas estaticas (que no se desplazan por el mapa, y por ende no necesitan tener informacion sobre todas las direcciones), lo que no implica que no puedan ser animadas. Una Decoration tiene dentro de si un sprite, que puede o no ser animado (segun la cantidad de cuadros que tenga).

Como se mencionó antes, ningun objeto renderizable conoce verdaderamente su posición ni su tamaño en pixeles, sino que se trabaja con unidades arbitrarias del juego. Así, las posiciones de las Decorations están expresadas en *tiles*, y los *offsets* de todos los objetos renderizables estan expresados con una cierta granularidad en terminos de tiles (el estandar es centesimas de tile, pero esto puede cambiarse en la configuracion del engine). Este offset se utiliza para desplazar la posición de renderización de los objetos respecto del origen del tile (esquina superior izquierda), y es útil a la hora diferentes objetos que comparten una posición (como el cuerpo y la cabeza de un personaje).
Cualquier objeto que herede de RenderizableObject puede ser renderizado por la Camara, que es el objeto encargado de traducir las unidades arbitrarias de juego en posiciones absolutas en pixeles dentro de la ventana para renderizar.

3.3. Sincronizacion y flujo de ejecucion

El flujo de ejecución del cliente (cambio entre vistas y sincronización con los mensajes del servidor) está orquestrado por el objeto GameStateMonitor. Este monitor encapsula un estado del juego (que es único para cualquier momento dado), en base al cual se determina cual es la siguiente vista que se debe mostrar.

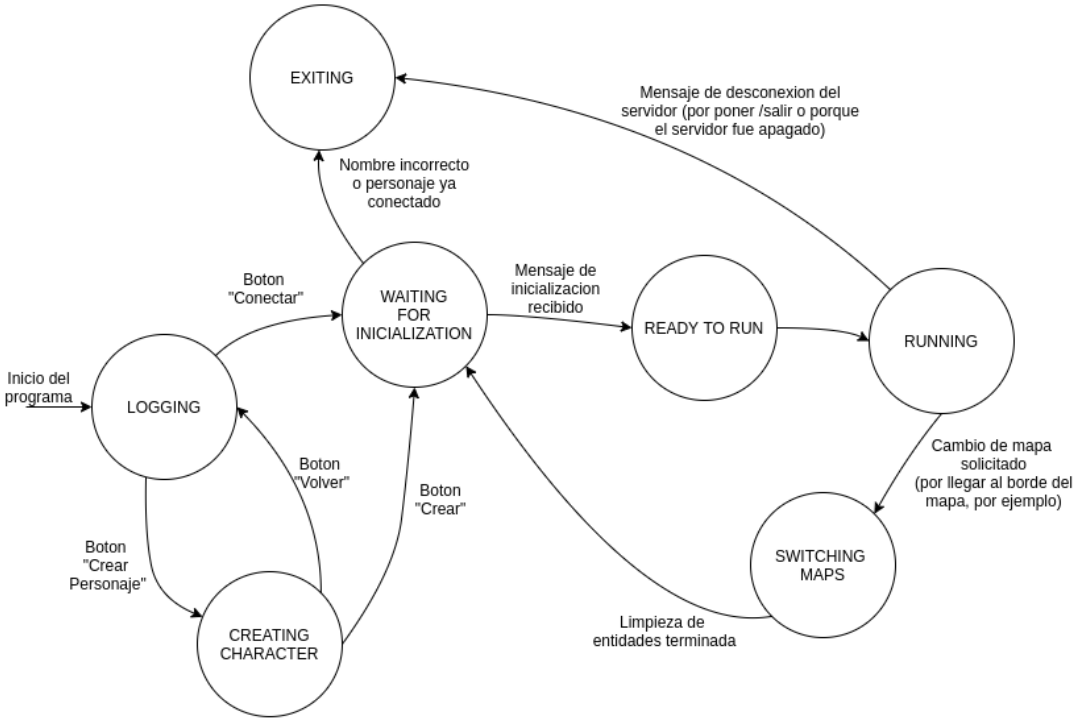

Figura 2: Cliente representado como maquina de estados finitos.

3.3.1. Estado LOGGING

Este estado del cliente es el primer estado que se establece al ejecutar el cliente. Indica que la vista que debe ser renderizada es la vista de inicio de sesión.

3.3.2. Estado WAITING_FOR_INICIALIZATION

Al iniciar sesión o cambiar de mapa, el cliente se pone en estado WAITING_FOR_INICIALIZATION, lo cual lo "bloquea"hasta que reciba el mensaje de inicialización del servidor. Como mecanismo de protección, si el servidor envia un mensaje de inicialización sin que el cliente esté en este estado, el thread de handleo de eventos provenientes del servidor se bloquea hasta que el cliente ingrese en este estado (Esto se debe a que, al cambiar de mapa, todas las entidades se borran de memoria, y se inicializa el nuevo estado del juego desde cero. Si el mensaje de inicialización llegara antes de que las entidades se terminen de borrar por completo, la entidad propia del jugador podría ser accidentalmente borrada instantáneamente luego de ser agregada).

3.3.3. Estado READY_TO_RUN

Indica que el mensaje de inicialización fue recibido correctamente, y que el juego se encuentra listo para empezar a correr en su estado jugable.

3.3.4. Estado RUNNING

Indica que el juego se encuentra corriendo en este momento, y es el estado que se tiene en todo momento en el que el jugador esté jugando propiamente el juego.

3.3.5. Estado SWITCHING_MAPS

Indica que el jugador solicito un cambio de mapa, y que por tanto se debe frenar la ejecución actual, borrar todas las entidades, y esperar un nuevo mensaje de inicialización por parte del servidor para el nuevo mapa.

3.3.6. Estado EXITING

Indica que, por alguna u otra razón, se está terminando la ejecución del cliente. Esto permite salir ordenadamente del juego.

3.3.7. Estado CREATING_CHARACTER

Estado que indica que se solicitó ir a la pantalla de creación de personaje.

4. Servidor

El servidor fue construido para funcionar de forma concurrente para varios jugadores que estuviesen conectados en múltiples mapas al mismo tiempo, para ello se dispuso a construir la parte lógica del juego para que corriera de forma independiente y se construyó el sistema de comunicación con los clientes por encima de ésta.

4.1. Lógica del Juego

El *GameLoop* es un Thread que actualiza todos los mapas (y cada uno de estos sus miembros que requieran actualizarse) cada un intervalo fijo de tiempo.

4.2. Eventos entrantes

El Dispatcher es un Thread encargado de obtener todos los eventos que deber realizar el servidor y despacharlo al objeto que haga el manejo correspondiente. Los objetos que permiten realizar el *"handle"* de un evento son hijos de la clase EventHandler, a partir de la cual también hereda la versión *Threaded* de la misma (BlockingThEventHandler).

4.3. Polling de cada mapa

Por cada mapa el servidor cuenta con un objeto Session que mantiene la comunicación del servidor con cada mapa dado. Por cada una de estas sesiones se tiene un Broadcaster (permite enviar a todos los jugadores conectados a la sesión un mismo evento) y un Observer que es el encargado de realizar el polling del mapa cada un intervalo fijo de tiempo para enviar la información de cambios del mapa a los clientes.

5. Comunicación

Una vez ejecutado el servidor, éste estará esperando conexión de nuevos clientes, que pueden iniciar sesión de dos formas diferentes: creando un jugador o entrando con un jugador existente (ver los mensajes correspondientes a continuación). Cuando un cliente ya está conectado y su jugador fuera agregado al mapa, recibirá el mensaje para inicializar el mapa seguido de un mensaje que actualiza todas las entidades y objetos tirados del mismo. Una vez hecho esto, el servidor le envía periódicamente mensajes que actualizan las posiciones de las entidades y cada algunas de estas también se envía información de las entidades y objetos caídos en el mapa. A continuación también se describen los mensajes para desconexión de un cliente.

5.1. Cliente → Servidor

Para al comenzar el cliente, iniciar sesión con un personaje nuevo se envía la siguiente información.

```
{
    "ev_id": EV_ID_CREATE,
    "client_id": ClientId,
    "name": string,
    "class_type": class_type_t,
    "race_type": race_type_t
}
```

Para iniciar sesión con un personaje ya existente, el cliente envía el siguiente mensaje al servidor.

```
{
    "ev_id": EV_ID_CONNECT,
    "client_id": ClientId,
    "player": {
        "name": string,
        "password": string // No utilizado
    }
}
```

Al presionar o levantar una de las teclas de movimiento, el cliente envía la información correspondiente en el siguiente mensaje.

```
{
    "ev_id": EV_ID_MOVE,
    "client_id": ClientId,
    "movement": {
        "action": mov_action_t,
        "direction": direction_t
    }
}
```

Al presionar la tecla para ataque (control), el cliente envía la siguiente información al servidor.

```
{
    "ev_id": EV_ID_ATTACK
    "client_id": ClientId,
}
```

Al presionar la tecla 'a' (agarrar) el cliente envía lo siguiente al servidor.

```
{
    "ev_id": EV_ID_PICKUP_LOOT
    "client_id": ClientId,
}
```

Al hacer click derecho sobre una posición del inventario, el cliente envía la siguiente información al servidor.

```
{
    "ev_id": EV_ID_DROP_LOOT,
    "client_id": ClientId,
    "slot": SlotId
}
```

Al hacer doble click (izquierdo) sobre una posición del inventario, el cliente envía la siguiente información al servidor.

```
{
    "ev_id": EV_ID_INVENTORY,
    "client_id": ClientId,
    "slot": SlotId
}
```

Al escribir cualquier mensaje en la consola, se envía la información dispuesta a continuación. Notar que se envía información de dónde se hizo click (izquierdo) sobre el mapa por última vez y qué *"Slot"* del inventario se hizo click (izquierdo) por última vez.

```
{
    "ev_id": EV_ID_COMMAND,
    "client_id": ClientId,
    "msg": string,
    "slot": SlotId,
    "target": {
        "x": int,
        "y": int
    }
}
```

Cuando un cliente se quiere desconectar y notificar al servidor de ello, envía el siguiente mensaje.

```
{
    "ev_id": EV_ID_DISCONNECT
    "client_id": ClientId,
}
```

5.2. Servidor → Cliente

Mensaje de inicialización de un mapa. Este se envía cada vez que se agrega un jugador a un mapa, esto quiere decir que sucede cada vez que un cliente inicia sesión (o crea un personaje) y cuando un jugador cambia de mapa.

```
{
    "ev_id": EV_ID_INITIALIZE_MAP,
    "client_id": ClientId, // Siempre será 0
    "map_info": {
        // map_info: Ver "mapas"
    },
    "player": {
        // player_init_data
        //
        // player_entity_data (Ver entity_data)
        //
        "pos": {
            "x": int,
            "y": int
        },
        "inventory": {
            //
```

```
            // inventory_data
            //
        }
    }
}
```

Mensaje de actualización gráfica de las entidades.

```
{
    "ev_id": EV_ID_UPDATE_ENTITIES,
    "client_id": ClientId, // Siempre será 0
    "entities": [
        {
            //
            // entity_data
            //
            "entity_id": EntityId,
            "name": string,
            "direction": direction_t,
            "move_speed": uint,
            "type_id": entity_type_t ,
            "curr_hp": uint,
            "max_hp": uint,
            "curr_mp": uint,
            "max_mp": uint,
            //
            // entity_player_data
            //
            "type_id": PLAYER,
            "head_id": uint,
            "body_id": uint,
            "helmet_id": uint,
            "armor_id": uint,
            "shield_id": uint,
            "weapon_id": uint,
            "curr_level": uint,
            "curr_exp": uint,
            "limit_exp": uint,
            //
            // entity_monster_data
            //
            "type_id": MONSTER,
            "sprite_id": uint
        },
        ...
    ]
}
```

Mensaje de actualización del *"loot"*, es decir, los objetos que son dejados en el piso del mapa.

```
{
    "ev_id": EV_ID_UPDATE_LOOT,
    "client_id": ClientId, // Siempre será 0
    "items": [
        {
```

```
            //
            // item_data
            //
        },
        ...
    ]
}
```

Mensaje de actualización de posiciones de entidades sobre el mapa, es enviado a cada paso de actualización del *Observer*.

```
{
    "ev_id": EV_ID_UPDATE_MAP,
    "client_id": ClientId, // Siempre será 0
    "positions": [
        {
            "entity_id": EntityId,
            "x": int,
            "y": int
        },
        ...
    ]
}
```

Mensajes para cuando un jugador hace daño a un objetivo o recibe daño de otra entidad.

```
{
    "ev_id": EV_ID_DEALT_DAMAGE,
    "client_id": ClientId, // Siempre será 0
    "dmg": int,
    "to": EntityId
}

{
    "ev_id": EV_ID_RECEIVED_DAMAGE,
    "client_id": ClientId, // Siempre será 0
    "dmg": uint
}
```

Mensaje del chat o consola para el cliente.

```
{
    "ev_id": EV_ID_CHAT_MESSAGE,
    "client_id": ClientId, // Siempre será 0
    "msg": string
}
```

Mensaje de actualización de inventario para un cliente. Este mensaje es privado, es decir, se envía únicamente al cliente controlando al jugador propietario del inventario.

```
{
    "ev_id": EV_ID_INVENTORY_UPDATE,
    "client_id": ClientId, // Siempre será 0
    "inventory": {
        "curr_gold": uint,
        "items": [
```

```
            {
                // 1era posición del inventario.
                "type": item_type_t
                //
                // Si type != TYPE_INVALID
                // item_data
                //
                "actual_stack": uint,
                "item_id": ItemId,
                "name": string,
                //
                // Ej: TYPE_POTION
                //
                "potion_info": {
                    "health_var": short,
                    "mana_var": short
                },
            },
            {
                // 2da posición del inventario.
                "type": item_type_t
            },
                ...
                {
                // 12ava posicion del inventario.
                "type": item_type_t
            }
        ]
    }
}
```

Mensaje para notificar a un cliente que se enviará un mapa nuevo, esto sucede cuando un jugador es cambiado de mapa.

```
{
    "ev_id": EV_ID_NOTIFY_NEW_MAP,
    "client_id": ClientId // Siempre será 0
}
```

Mensaje informativo a un cliente para notificar que ha sido desconectado del servidor.

```
{
    "ev_id": EV_ID_DROP_CLIENT,
    "client_id": ClientId // Siempre será 0
}
```