Universidad de Buenos Aires

Facultad de Ingeniería

75.42 - Taller de Programación

Lenguaje: C++

Primer cuatrimestre de 2022

Manual Técnico

Burgos, Juan Sebastian	100113	jsburgos@fi.uba.ar
Chávez Cabanillas, José E.	96467	jchavez@fi.uba.ar
Del Pup, Tomás	102174	tdelpup@fi.uba.ar
Huzan, Hugo	67910	hhuzan@fi.uba.ar

Índice

1. Introducción

El presente trabajo realizado se basa en un remake del juego de estrategía Dune 2000, desarrollado en el año de 1998, este juego se basa en la conquista de un planeta llamado Dune, por parte de tres casa, "Harkonnen, Ordos y Atreides", el juego original contaba con un campaña de un solo jugador y un multijugador online, en este proyecto solo se recrear el multijugador.

2. Requerimientos del Sistema

El proyecto usa varias librerias para poder ejecutarse, así como también de un sistema operativos basado en alguna distribución GNU/Linux, a continuaciíon se mencionaran las librerias necesarias para poder compilar y ejecutar el juego.

2.1. SDL 2

Esta librería es el motor gráfico del juego, se encarga de todo lo relacionado con la renderización, musicalización e interacción con el usuario (Cliente), determinando cada acción del usuario para luego mostrarla en pantalla.

A su vez esta librería cuenta con librerias adicionales, que complementan y forman parte de toda el motor gráfico. Estas librerías son las de SDL2 Mixer y SDL2 TTF, encargandose de la reproducción de la música del juego, y la visualización de fuentes respectivamente.

Figura 1: SDL

2.2. Qt5

Qt5, es una librería para trabajar con interfaz gráfica, la ventaja de esta librería es que utiliza el lenguaje de programación `C++` de forma nativa, lo que cual fue una gran ventaja para el proyecto, cuenta con diversos módulos.

Dentro de estos módulos, para el proyecto se requirieron el moódulo de Widgets y de Multimedia, siendo el primero para la creación de la interfaz, mediante el uso de buttons, labels, lineEdits, etc. El segundo, se encarga de la música de fondo entre los diversos menús del cliente.

Cabe resaltar que Qt se empleó también para la creación del editor de mapas del proyecto, dandonós un uso más a esta librería.

Figura 2: Qt

2.3. YAML-cpp

El nombre de la libreria nos da una pista sobre el uso de esta, su funcionalidad radica en el parser de archivos YAML, el cual se usa tanto para cargar los mapas asi como también para la carga de configuración del juego.

3. Descripción General

En un principio, teníamos la idea de que cada cliente pueda transmitirle sus movimientos al servidor para retransmitirle al resto de los usuarios ese movimiento. Es por esto que requerimos un modelo autoritario del lado del servidor. El servidor es quien simula el juego, mientras que los usuarios solo le pueden comunicar sus intenciones al servidor, que luego corre la simulación y genera la respuesta apropiada, dejando a todos los jugadores sincronizados. Los clientes, a traves de sus movimientos (jugadas), envían por el socket un mensaje siguiendo un protocolo, que luego un handler del lado del servidor toma y encola el comando que simule lo que el usuario espera. Los comandos son sacados de la cola de manera FIFO por el Engine, quien es el encargado de tomar estos comandos, ejecutarlos y enviar las respuestas apropiadas a los usuarios. El motor del juego simula el paso del tiempo para aquellas partes del modelo que lo requieran. Estas notificaciones son luego recibidas por el usuario para actualizar su representacion visual del mapa, sea moviendo sprites de lugar o reproduciendo sonidos como respuesta a una acción.

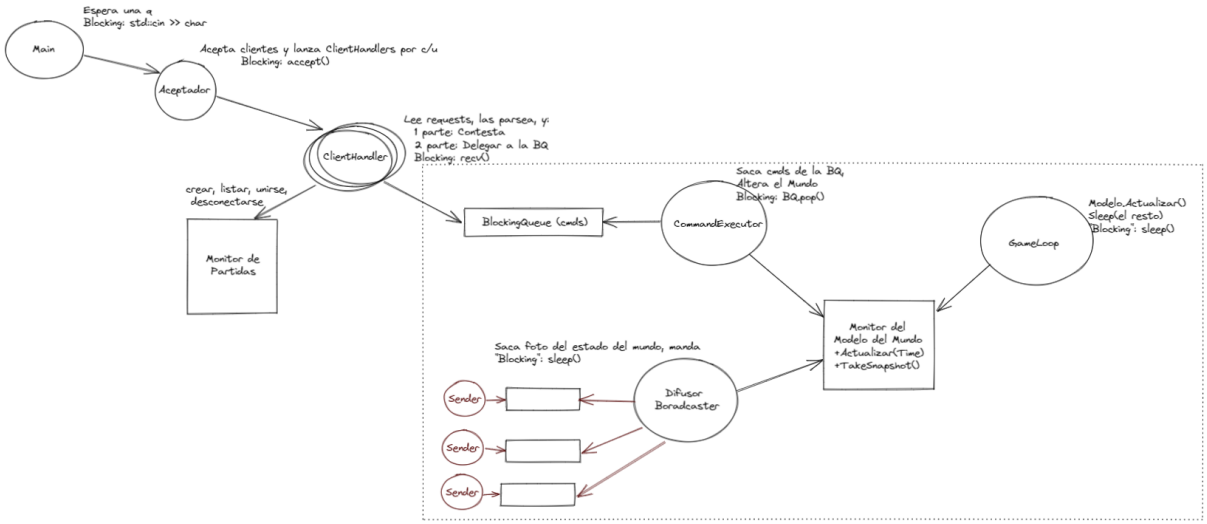

Figura 3: Módelo del server

4. Descripicón de Clases

En esta sección hablaremos de algunas de las clases implementadas para la resolución del proyecto, algunas de las características que estas tienen y partes de su implementación.

4.1. Cliente

El cliente se ocupa mayoritariamente de enviar, recibir y actuar ante comandos emitidos por el servidor. Posee varias clases que se ocupan de organizar, distribuir y renderizar la información que administra.

Este consta de dos partes principales, la primera como ya se hizo mención anteriormente (Qt5), es la interfaz de conexíon al servidor, creación, unión y listado de partidas que se encuentren en el servidor. La segunda parte es el renderizado del juego, del cual se encarga SDL, y donde suceden la mayoria de comandos tanto emitidos, como recibidos entre el cliente y el servidor.

4.2. Renderizado

Para el renderizado se precargaron las imagenes de todos los elementos requeridos, unidades, edificios, terreno, gusano.

Estos son dibujados de acuerdo a los eventos que ocurran en la partida, por defecto al seleccionar un mapa a la hora de la creación de la partida, el server envía el contenido del archivo, para su renderizado, este contiene una matriz con cada uno de los tiles que conforman el terreno del juego.

Los eventos determinan la animación a realizarse, ya sea la construcción de un edificio, o el movimiento de una unidad.

4.3. Esquema General

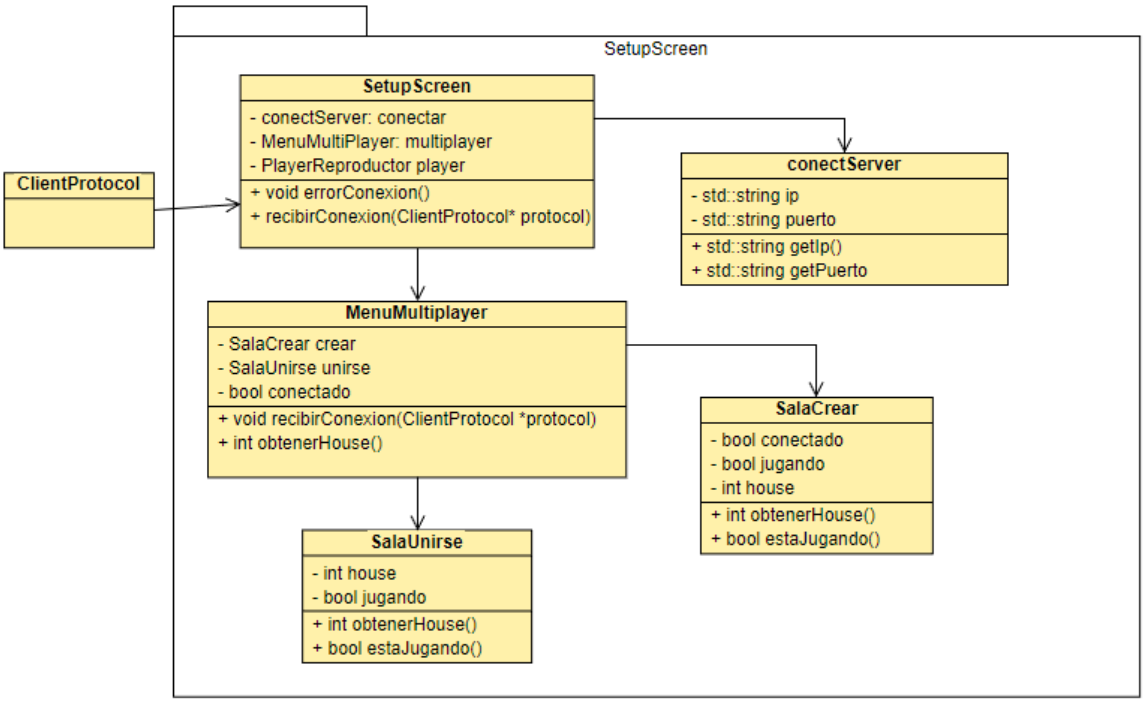

Figura 4: Clase SetupScreen: interfaz del login

La ventana de Login, generada con Qt permite que el usuario pueda ingresar la ip y puerto necesarios para conectarse al servidor. La misma le permite al usuario elegir una partida de las disponibles o crear la suya. Al introducir los datos, el Login se intenta conectar al servidor y cuando se conecta, puede recibir una lista de partidas vigentes a las que conectarse, asi como las demás operaciones ya mencionadas.

4.4. Audio y Mapa

4.4.1. Mapa - Algoritmo A-Star

El mapa del juego al ser este desde una perspectiva aérea, es una ventaja bastante importante al renderizarlo, sin embargo lo importante no fue el renderizado en si del mapa, si detectar el tipo de terreno que este contenia para poder realizar movimientos sobre el mismo, para esto se emplo el algoritmo A-Star, el cual determina el recorrido, con la solvedad de que necesita el mapa para poder determinar dicho recorrido.

El algoritmo A-Star en si es bastante sencillo, no obstante se tiene que tener en cuenta que unidades, enemigos lo van a usar

```
for (int add_x = -1; add_x <= 1; add_x++) {
    for (int add_y = -1; add_y <= 1; add_y++) {
        std::pair<int, int> neighbour(i + add_x, j + add_y);
        if (isValid(map, neighbour)) {
            if (isDestination(neighbour, dest)) {
                cellDetails[neighbour.first][neighbour.second].parent
                            = {i, j};
                path = tracePath(cellDetails, dest);
                return path;
            }
            if (!closedList[neighbour.first][neighbour.second] &&
                isUnblocked(map, neighbour, isVehicle, isWorm)) {
                double gNew, hNew, fNew;
                gNew = cellDetails[i][j].g + 1.0;
                hNew = calculateHValue(neighbour, dest);
                fNew = gNew + hNew;
                if (cellDetails[neighbour.first][neighbour.second].f
                    == -1 || cellDetails[neighbour.first]
                    [neighbour.second].f > fNew) {
                    openList.emplace(fNew, neighbour.first,
                                     neighbour.second);
                    cellDetails[neighbour.first][neighbour.second].g
                                = gNew;
                    cellDetails[neighbour.first][neighbour.second].h
                                = hNew;
                    cellDetails[neighbour.first][neighbour.second].f
                                = fNew;
                    cellDetails[neighbour.first][neighbour.
                                second].parent = {i, j};
                }
            }
        }
    }
```

```
}
```

Código 1: Método: A-Star

Este algoritmo dentro del juego es consultado, siempre que se quiera mover cualquier unidad, de los distintos jugadores para asi poder enviar el camino que este debe seguir posición a posición, en cada tick del server, al cliente.

4.4.2. Audio

Este juego contiene audios muy variados, dependiendo el tipo de evento que se va a realizar, los audios del juego estan precargados dependiendo el tipo de Casa que elijes, estos audios son seleccionados dependiendo si es un evento general del juego o es un evento individual, como seleccionar o mover una unidad.

El motor del juego, determina el momento en el que debe reproducirse el audio, de acuerdo a los eventos realizados por el cliente y si el servidor valida el evento que se produce. Si solo se selecciona una unidad, el evento no deber ser consultado por al servidor, simplemente se reproduce el audio.

4.5. Editor

Lo primero a destacar que el editor de mapas del juego se creo exclusivamente con la libreria de Qt, además de emplear la librerias de parser de YAML, para poder crear el archivo del mapa.

El editor en su mayor parte cuenta con dos funciones principales, crear un mapa o editarlo. Estas funcionalidades esta muy ligadas entre si dado que usan las mismas clases para dibujar y renderizar correctamente cada tile del terreno, para esto usa los eventos provenientes del mouse, ademas de las funciones propias de clase las cuales permiten hacer drag o drop entre ellas, usando el mouse para detectar el evento.

El evento drop es el responsable de distinguir mediante un tag, además de controlar la posición donde el tile de terreno se va a dibujar.

```
if (event->mimeData()->hasFormat("image/x-map-piece") &&
    findPiece(targetSquare(event->pos())) == -1) {
        QByteArray pieceData = event->mimeData()->data(
                                "image/x-map-piece");
        QDataStream dataStream(&pieceData, QIODevice::ReadOnly);
        Piece piece;
        piece.rect = targetSquare(event->pos());
        dataStream >> piece.pixmap >> piece.location >> piece.codigo;
        int codigo = piece.codigo;
        if (agregarCodigoAListas(codigo, piece.rect.y(),
                                        piece.rect.x())) {
            pieces.append(piece);
            highlightedRect = QRect();
            update(piece.rect);
            event->setDropAction(Qt::MoveAction);
            event->accept();
        } else {
            highlightedRect = QRect();
            event->ignore();
        }
} else {
    highlightedRect = QRect();
```

```
    event->ignore();
}
```

Código 2: Método: dropEvent

Después de tener el mapa dibujado, el editor hace una comprobación de los datos antes de guardar el mapa. Para generar el archivo YAML, emplea el parser, y genera un YAML::Emitter, donde se carga la información para después ser guardada.

A continuacón se muestra una imagen de la interaccion entre las distintas clases que se emplearon para la elaboración del editor, la misma representa los atributos más importantes, así como tambien los métodos.

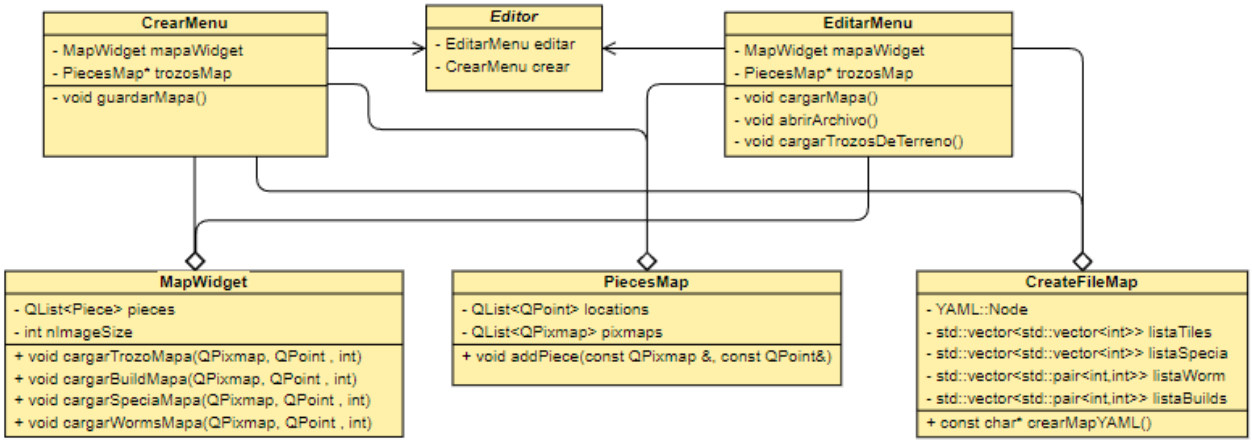

Figura 5: Diagrama de clases del editor

4.6. Server

El servidor es quien tiene mucha influencia en lo que respecta a flujo de juego, al tener un módelo autoritario, es quien decide que acciones son validas o no a traves de los mensajes que llegan desde el cliente, los cuales son almacenados en una cola bloqueante. Estos mensajes son procesados en orden, y dependiendo la operación realizada se ejecuta dentro del modelo de juego que cada partida tiene.

4.6.1. Modelo de Juego

El modelo, es quien se encarga de almacenar todos los datos de una partida de juego, ID de jugadores, unidades que tiene cadad jugador, edificios gusanos, este modelo, es a tiempo real, se ajusta al tiempo real del juego para una mejor simulación.

5. Descripción de Archivos y Protocolos

5.1. Archivos

Para el juego, se emplearon archivos tipo YAML para la configuración, este archivo contiene la información de las unidades (daño, vida, velocidad, etc), edificios (dimensiones, vida), tipos de armas (daño, alcance). El cual puede ser modificado a mano. Por otro lado tenemos los archivos que guardan información sobre el mapa, estos archivos pueden tener cualquier nombre.

5.2. Protocolo

Tanto el cliente como el server poseen un archivo clientprotocol.h y serverprotocol.h que heredan de una clase protocol.h que posee una lista de defines importantes utilizados para facilitar el manejo de opcodes de todo tipo. Entre las más importanes están:

- PROT_UNIRSE_CODE: Representa el mensaje de unirse a una partida.
- PROT_LISTAR_JUEGOS_CODE: Este operación lista todas las partidas activas del server, asi como también la cantidad de jugadores en la partida.
- PROT_CREAR_PARTIDA_CODE: Crea una partida, con un número determinado de jugadores, asi como también el nombre de la partida, y la casa con la que deseas jugar la partida.
- PROT_LISTAR_MAPAS_CODE: Lista los mapas disponibles, estos mapas se encuentran en la carpeta de mapas del juego, esta lista contiene el nombre del mapa, asi como la cantida de jugadores que se pueden jugar en el.
- PROT_MOVER_CODE: Envia la operación de mover a una o varias unidades, desde una posicion inicial, hasta su posición final.
- PROT_CONSTRUIR_CODE: Envia la operación de construir un edificio, empleando el codigo del mismo, asi como la posicion en la que se desea construir.
- PROT_CREAR_UNIDAD_CODE: Envía la petición de creación de una unida, determinada por el código de la unidad.
- PROT_ATACAR_CODE: Envía la operación de atacar a una unidad, usando los códigos de unidad, el ID del jugador y la posición de la unidad atacada.
- PROT_DESTRUIR_CODE: Envía la solicitud de destrucción de un edificio, empleando el codigo de edificio asi como el ID del jugador.