U.B.A. FACULTAD DE INGENIERÍA

Departamento de Computación

[75.42/95.08] - Taller de Programación I

2022

Primer Cuatrimestre

Dune 2000 - Documentación Técnica

APELLIDO, Nombres	N° PADRÓN	Carrera
SABELLA ROSA, Cristóbal	106440	Ingeniería en Informática
GADDI, María Pilar	105682	Ingeniería en Informática
SARDELLA, Florencia	105717	Ingeniería en Informática

1. Índice

2. Servidor

2. 1 Descripción general

El servidor es el encargado de conectarse e interactuar con los diferentes clientes que se unan a una partida. También, obtiene la información de todos los mapas disponibles que le provee el editor para que luego los clientes puedan recibirlos y dibujarlos.

2. 2 Esquema de Threads

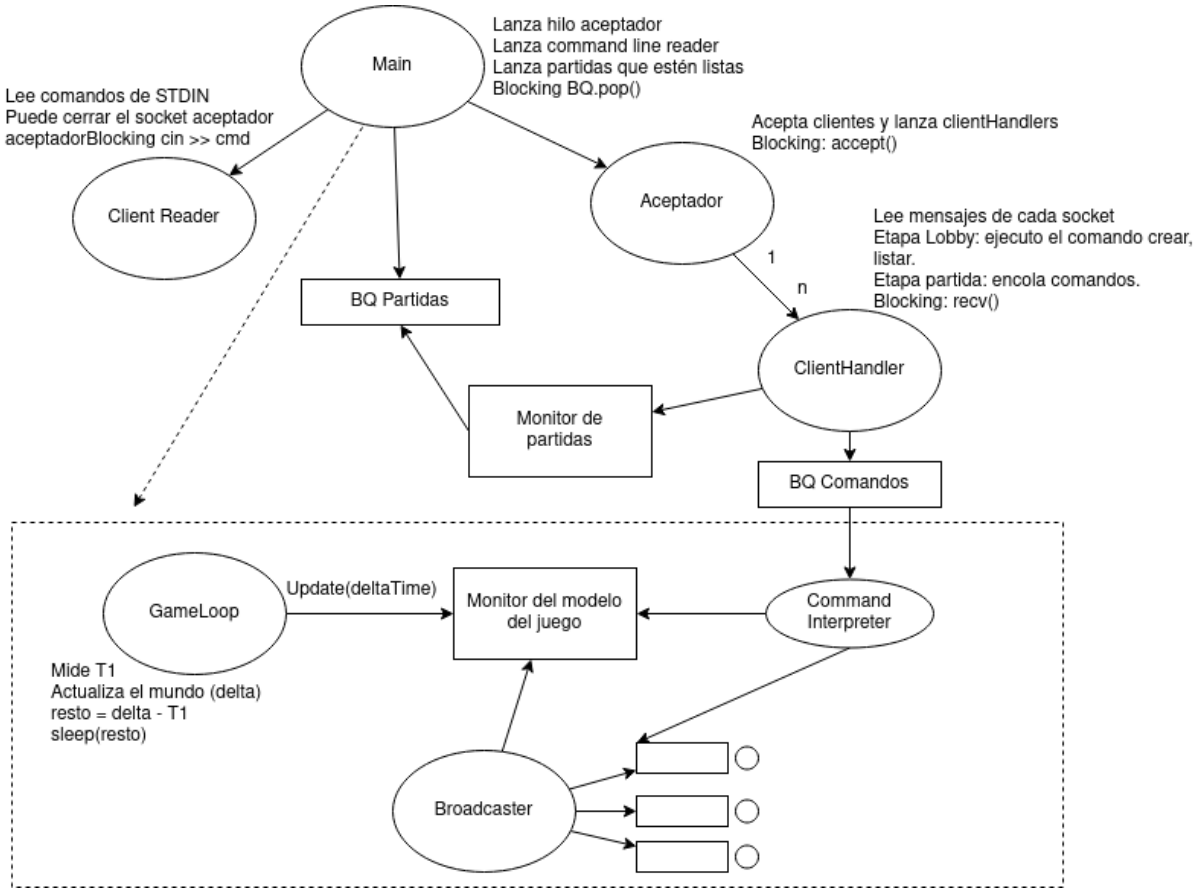

Imagen 1: modelo de threads del Server

3. Cliente

3. 1 Descripción general

El cliente es el encargado de interactuar con el jugador a partir de una interfaz gráfica. Por medio de una dirección de IP y de PORT, el cliente podrá establecer una conexión con un servidor que se encuentra escuchando en ese PORT. Luego, a partir de los datos que brinde el jugador por medio de la interfaz gráfica, se podrá establecer una comunicación cliente-servidor para darle cierta jugabilidad al proyecto.

3. 2 Esquema de Threads

A continuación, se listan los threads utilizados en el cliente:

- **Drawer:** es el hilo que se encarga de dibujar durante la partida. Cada vez que el mismo recibe algo para dibujar lo va a pegar al mapa.
- **Server Receiver:** es el hilo que recibe los mensajes que van a ser enviados por el servidor.
- **User Input Receiver:** es el hilo que recibe las operaciones que quiere realizar el jugador durante toda la partida. Estas operaciones luego son interpretadas por el server dispatcher para que pueda comunicarse con el servidor y que este último aplique su lógica de juego.
  Este hilo va a pushear los mensajes recibidos en una cola bloqueante.
- **Server Dispatcher:** es el hilo que manda mensajes al servidor. Dependiendo de la operación recibida, enviará distintos mensajes.
  Este hilo va a popear los mensajes que se encuentran en la cola bloqueante (de esta forma es como recibe el número de operación) y le enviará el mensaje al servidor.
- **Game Waiter:** es el hilo que espera a que se obtenga la cantidad de jugadores requeridos para que cada jugador pueda salir de la pantalla de espera e ir al comienzo del juego.

3.3 Diagrama de clases

En este primer diagrama de clases se pueden ver los diferentes threads y sus relaciones. El ServerDispatcher popea todo lo que pushea el UserInputReceiver en una cola bloqueante del tipo ClientInput. En esta clase se almacena toda la información que recibe UserInputReceiver del jugador. Por lo tanto, una vez que ServerDispatcher obtiene los datos correspondientes, le envía al servidor los respectivos mensajes.

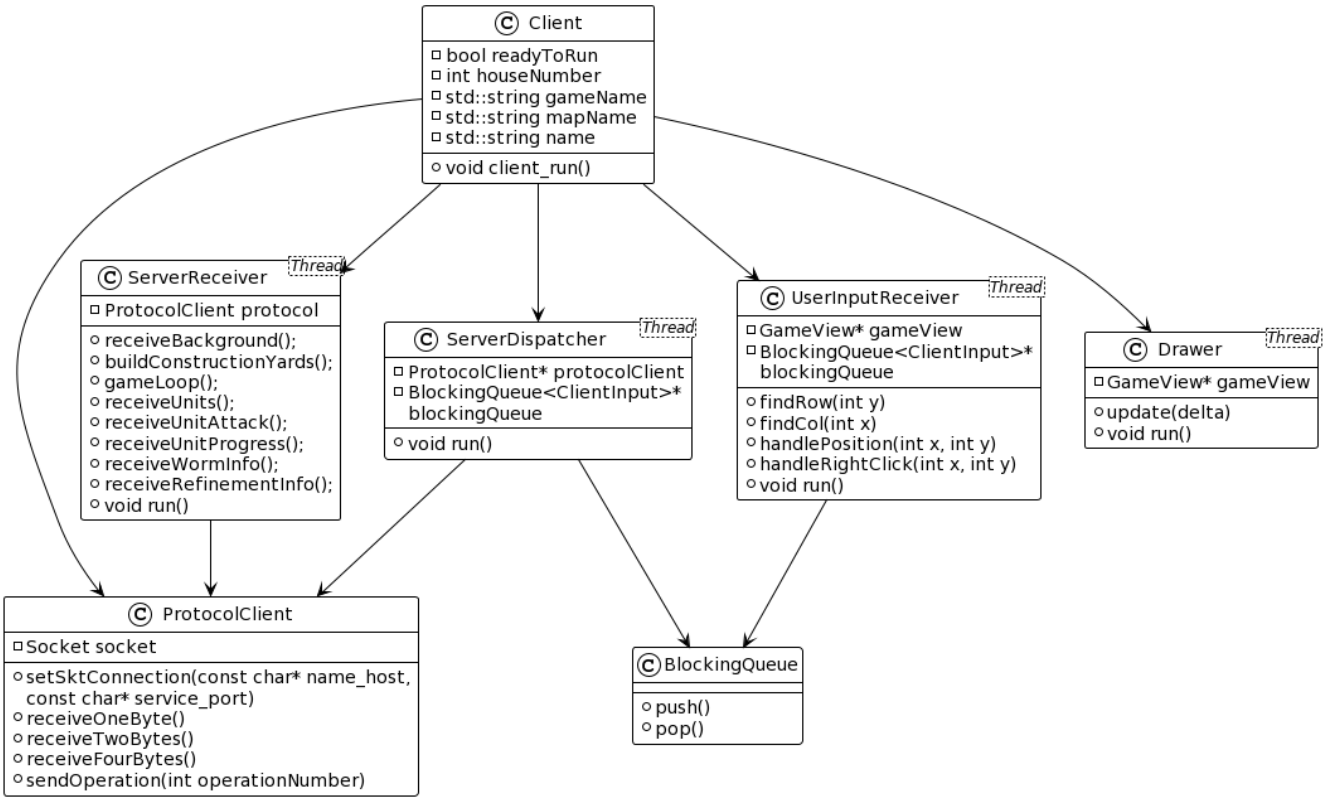

Imagen 2: Diagrama de clases con modelo de threads del client

En el segundo diagrama de clases se puede ver más en detalle la clase Drawer, y como este hilo se relaciona con el GameView y el MapView.
El hilo drawer renderiza el GameView y actualiza la vista constantemente. Cabe mencionar que en el caso del GameView se tienen recursos compartidos entre distintos hilos. Por lo tanto, para que no se generen problemas de concurrencia, se optó por la utilización de un monitor.
La clase GameView es la encargada de mostrar toda la vista del juego, tanto el mapa (por medio de la clase MapView) como la ventana (por medio de la clase SdlWindow). Sin embargo, para que no se este renderizando toda la vista completa constantemente, se tiene la clase Camera que únicamente va a renderizar un elemento cuando este mismo se encuentre dentro del rango de visión que posea en ese momento el jugador.

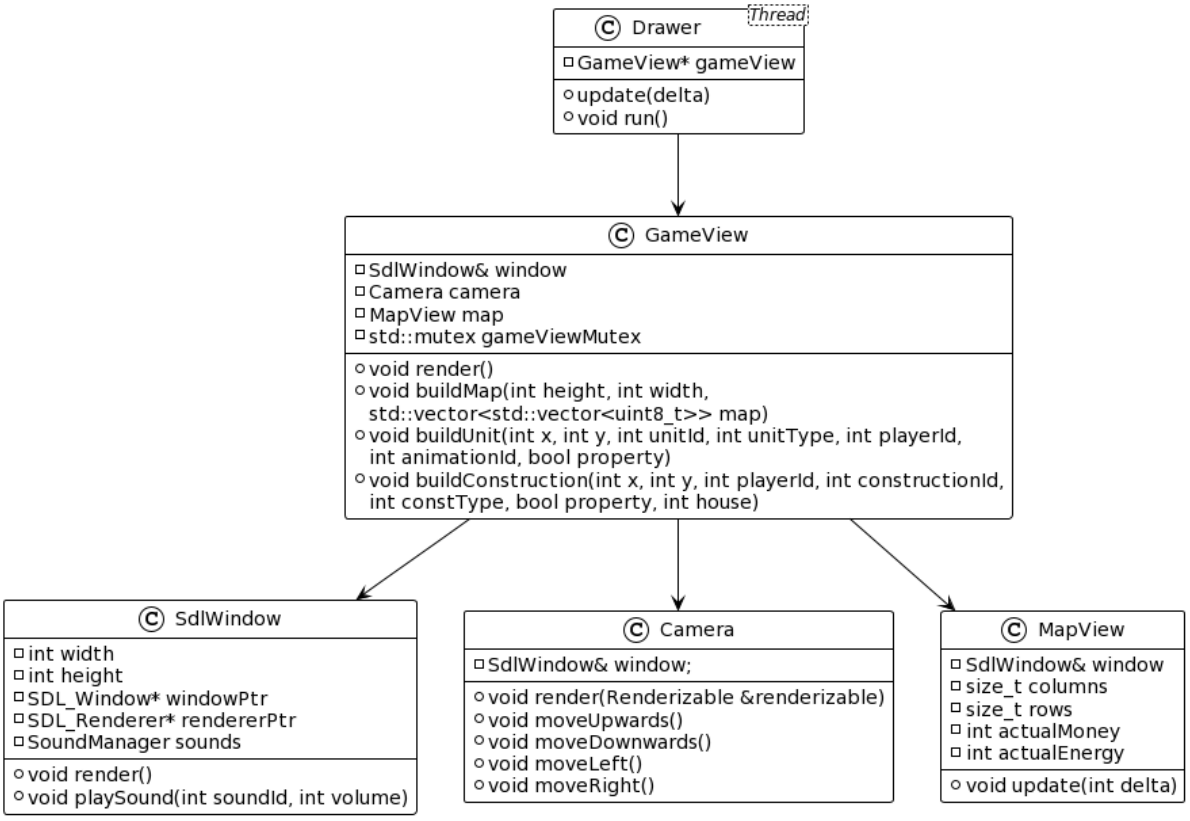

Imagen 3: Diagrama de clases con modelo de interfaz gráfica

4. Editor

4. 1 Descripción general

El editor es el encargado de llevar a cabo la edición (creación y modificación) de los mapas. El mismo permite también guardarlos en formato YAML. De esta forma, el servidor los podrá cargar y utilizar a la hora de comenzar una partida.

4. 2 Diagrama de clases

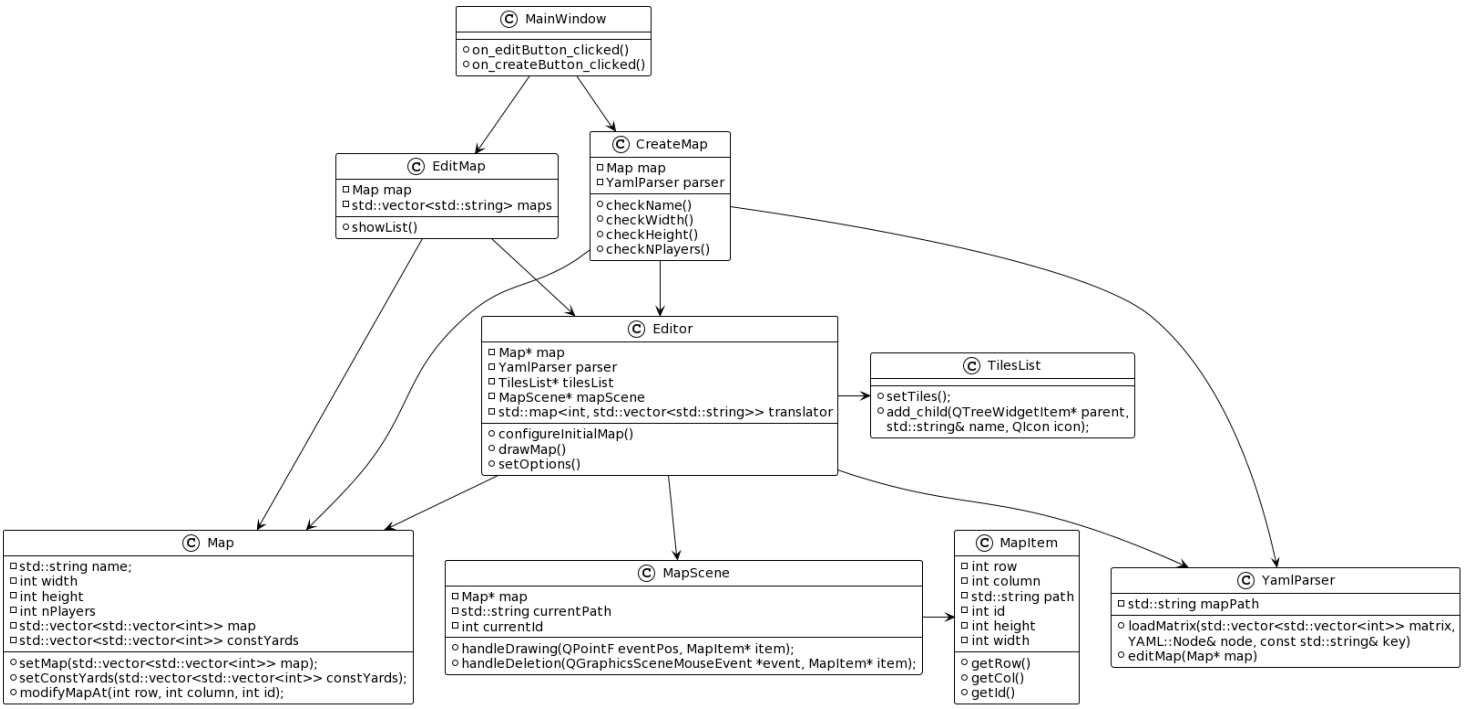

Imagen 4: Diagrama de clases con modelo del editor

5. Protocolo de comunicación

En cuanto al protocolo de comunicación, se llevó a cabo un protocolo binario. Este mismo sirve para poder establecer una comunicación cliente-servidor. En el caso del servidor, se tienen las siguientes operaciones que el mismo puede enviar al cliente y este último debe interpretar:

Operación 0: operación exitosa
Operación 1: operación fallida
Operación 2: broadcast de unidades
Operación 3: edificio construido
Operación 4: ataque a unidades
Operación 5: ataque a edificio
Operación 6: partida perdida (1 byte)
Operación 7: partida ganada (1 byte)
Operación 8: unidad en construcción
Operación 9: edificio en construcción
Operación 10: gusano de arena
Operación 11: refinamiento de especia
Operación 12: edificio destruido

Más allá de las operaciones que el cliente pueda recibir, se debe recibir primeramente el arranque de una partida, con toda su información:

Arrancar partida:

int8 **partidaListaParaEmpezar**
int16 **filasDeMapa**
int16 **columnasDeMapa**
int8 **mapa**
int8 **cantidadJugadores** (con ids implícitas, el id del Centro de construcción es el mismo que el id de su dueño o jugador).

int16 **largoDeNombre**
str **nombre**
int16 **posCentroDeConstruccionX**
int16 **posCentroDeConstruccionY**
int8 **numeroDeCasa**

Una vez que se arranca la partida, cada cliente va a estar constantemente recibiendo un broadcast de unidades, con toda la información de cada una de ellas.

Broadcast de unidades:

int8 **numeroDeOperación** (en este caso es la operación 2)
int16 **cantJugadores**
    int8 **idJugador**
    int32 **energíaJugador**
    int32 **dineroJugador**
    int16 **cantUnidades**
        int16 **posX**
        int16 **posY**
        int8 **tipoUnidad**
        int8 **direccion**
        int16 **idUnidad**

Durante el juego, se podrán construir edificios, entrenar unidades y atacar. Para ello, se tienen las siguientes operaciones:

Edificio en construcción:

Int8 **numeroDeOperación** (en este caso es la operación 9)
Int8 **tipoConstruccion**
int8 **porcentajeConstruccion**

Edificio construido:

Int8 **numeroDeOperación** (en este caso es la operación 3)
int8 **playerId**
int16 **idEdificio**
int8 **tipoEdificio**
int16 **x**
int16 **y**

En caso de que falle el posicionamiento de un edificio ya construido, no se obtendrá respuesta del lado del servidor y simplemente no se posicionará el edificio en tal posición.

Ataque a edificio:

Int8 **numeroDeOperación** (en este caso es la operación 5)
int16 **idAtacante**
int16 **idEdificio**
int16 **vidaEdificioActual**
int16 **vidaEdificioTotal**

Unidad siendo entrenada (en “construcción”):

Int8 **numeroDeOperación** (en este caso es la operación 8)
Int16 **cantidadUnidadesEnConstruccion**
    Int8 **idJugador**
    Int8 **tipoUnidad**
    Int8 **porcentajeConstruccion**

Como constantemente el servidor va a estar enviando un broadcast de unidades, cuando la unidad sea entrenada completamente se va a incorporar al broadcast para que el cliente la reciba.

Ataque a unidad

Int8 **numeroDeOperación** (en este caso es la operación 4)
int16 **idAtacante**
int16 **idAtacado**
int16 **vidaActual**
int16 **vidaTotal**

Luego, se agregaron los siguientes features: existencia de un gusano de arena que aparece de forma aleatoria en cualquier momento de la partida y refinerías para que la cosechadora pueda ir en busca de la especia Melange con el objetivo de que el jugador obtenga dinero al entregarla en las refinerías. De esta forma, se estableció que el servidor debe enviar los siguientes mensajes:

Refinamiento de especia:

Int8 **numeroDeOperación** (en este caso es la operación 11)
int16 **cantidadDeTilesDeEspecia**
    int16 **x**
    int16 **y**
    int8 **estadoDeEspecia**

Gusano de Arena:

Int8 **numeroDeOperación** (en este caso es la operación 10)
int16 **x**
int16 **y**

En el caso del cliente, se tienen las siguientes operaciones que el mismo puede enviar al servidor y este último debe interpretar:

Operación 1: Unirse a partida
Operación 2: Listar partidas
Operación 3: Crear partida
Operación 4: listar mapas
Operación 5: Crear unidad
Operación 6: Crear edificio
Operación 7: Atacar
Operación 8: Mover unidad
Operación 9: Posicionar edificio
Operación 10: Destruir edificio

Crear unidad
Int8 **numeroDeOperación** (en este caso es la operación 5)
Int8 **tipoUnidad**

Crear edificio
Int8 **numeroDeOperación** (en este caso es la operación 6)
Int8 **tipoEdificio**

Atacar a unidad o edificio
Int8 **numeroDeOperación** (en este caso es la operación 7)
int8 **unidad o edificio (0 unidad, 1 edificio)**
int16 **idAtacante**
int16 idAtacado

Mover unidad
Int8 **numeroDeOperación** (en este caso es la operación 8)
int16 **idUnidad**
int16 **x**
int16 **y**

Posicionar edificio
Int8 **numeroDeOperación** (en este caso es la operación 9)
int16 **x**
int16 **y**

El jugador podrá destruir un edificio propio por voluntad propia, para poder obtener un reintegro del mismo.

Destruir edificio
Int8 **numeroDeOperación** (en este caso es la operación 10)
int16 **idEdificio**