75:42 - Taller de Programación I

Ejercicio Nº ___ Padrón _____

Alumno _____ Firma _____

Nota:		**Corrige**:		**Entrega #1**
				Fecha de entrega
				Fecha de devolución

Nota:		**Corrige**:		**Entrega #2**
				Fecha de entrega
				Fecha de devolución

El presente trabajo, así como la entrega electrónica correspondiente al mismo, constituyen una obra de creación completamente personal, no habiendo sido inspirada ni siendo copia completa o parcial de ninguna fuente pública, privada, de otra persona o naturaleza.

Facultad de ingeniería de la UBA

75.42/95.08 Taller de Programación I
Cátedra Veiga

Trabajo Práctico Final
Portal

Primer cuatrimestre de 2019

Integrante	Padrón	Correo electrónico
Beroch, Santiago	101135	sberoch@gmail.com
Calvo, Mateo Iván	98290	mateocalvo@gmail.com

Link al repositorio:
https://github.com/sberoch/PortalTaller

Índice

1. Manual de Proyecto

1.1. División de tareas y evolución del proyecto

Aquí se presenta la división de tareas y el tiempo durante el cual se llevaron a cabo. Para consultar el cronograma propuesto, consultar el apéndice A. El cronograma real, al momento de la entrega, se presenta en el siguiente diagrama:

Semana	1	2	3	4	5	6
S: Draft del modelo - Prueba de concepto Box2D						
S: Escenario y elementos estáticos						
S: Chell y elementos dinámicos						
S: Portales, física completa, compuertas						
S: Multipartidas, victoria y derrota						
Cmp: Testing, correcciones, documentación						
Cmp: Tareas compartidas, interfaz cliente-servidor						
C: Mostrar imágenes, animaciones y moverlas						
C: Renderizado del escenario, cámara						
C: Animación de elementos dinámicos						
C: Finalización de la parte gráfica						
C: Música, sonido y pantallas de login						

Tabla 1: División real de las tareas del proyecto. Alumno responsable del cliente: Beroch, Santiago. Alumno responsable del servidor: Calvo, Mateo. Leyenda: S (servidor), Cmp (compartido), C (cliente).

1.2. Enunciado

El enunciado del trabajo práctico puede consultarse en el apéndice A.

1.3. Inconvenientes encontrados

A continuación se listan los inconvenientes encontrados durante el desarrollo del trabajo práctico.

1.3.1. En el servidor

- Al utilizar el proyecto *Testbed* provisto en `Box2D` para realizar pruebas de concepto: Se debió aprender a utilizar muy básicamente la utilidad *premake*, para poder compilar el proyecto. Además, se debió indagar

un poco en la implementación del mismo para realizar pruebas que involucraran lógica del juego dentro de la simulación de *Testbed*.

- En la implementación de las colisiones: en afán de lograr una jugabilidad *razonable* (en el sentido de ser divertida y a la vez sin sorpresas ni bugs), fue necesario considerar un sistema de colisiones que permita un filtrado de las mismas, para evitar eventos tales como la doble destrucción de una entidad como así también los ciclos infinitos de teletransportación. Si bien al momento de la entrega no está implementado el filtrado de colisiones, para evitar que un jugador se teletransporte indefinidamente se establecieron *hitboxes* de colisión para las superficies relevantes, que no abarcaran todo el bloque sino un área delgada centrada en cada arista de los bloques.

- En la utilización de la librería `Box2D`: Para lograr más fácilmente el polimorfismo entre entidades del juego, se decidió modificar la librería en la manera en que la misma mantiene una referencia a los datos del usuario: En lugar de un puntero `void*` se agregó un puntero `Colisionable*`. Lo anterior permitió procesar colisiones con menos código y sin utilizar casteos, de manera polimórfica.

1.3.2. En el cliente

- Se perdio mucho tiempo para la instalacion y utilizacion de `yaml-cpp` para levantar el escenario.

- SDL no fue particularmente problematico pero costo poder aprenderlo en las primeras semanas.

- Un gran problema fue el poder implementar un manejador de input del usuario polimorfico. Fue confuso por varias razones. Hay que tratar eventos de mouse diferente a los de teclado. Tambien se envian diferentes parametros segun el tipo de evento generado. Y hasta podria no generarse un evento que se envie al servidor (caso tecla para activar fullscreen). Finalmente se acerco a una solucion polimorfica, pero que tras algunos eventos hacia crashear al programa. Por todo esto, se termino utilizando un switch para manejar los inputs del usuario.

- Se intento emplear mas `shared pointers` sin exito, por lo que solo estan presentes para VistaObjeto.

- No se pudo generar un archivo de supresiones para no tener en cuenta los leaks de SDL, ya que se vuelve tan pesado que el programa no

puede terminar correctamente y por lo tanto no se pueden alcanzar todos los leaks de SDL. Por lo tanto, solo se pudo hacer una analisis "manual"para chequeos de perdida de memoria.

1.3.3. En la integración cliente-servidor

- En la creación del archivo de instalación: Al momento de la entrega, no fue posible compilar y linkear los ejecutables uniformemente en las dos computadoras en las que el proyecto fue desarrollado. Los problemas de compilación o linkeo parecen estar asociados a la manera en que se incluye la librería `yaml-cpp` en el proyecto.

1.4. Análisis de puntos pendientes

Al momento de la primera entrega, restan implementar las siguientes características:

1.4.1. En el cliente

- Pantallas de sala/seleccion de partida.
- Pulido de animaciones, ya que muchas de las animaciones intermedias (ej. pasar del estado corriendo al estado idle) no se contemplaron.
- Pulido de texturas
- Manejador de eventos polimorfico, en lugar del actual con switch

1.4.2. En el servidor

- Partidas multijugador, a través de la implementación de *Salas* que permitan unirse e iniciar una partida.
- Integración de los métodos de colisión, que no se encuentran en la entrega debido a los problemas mencionados en la sección anterior (destrucciones dobles, creaciones repetidas, etc).
- Implementación de la lógica de portales, a través de una clase *Arma* que mantenga una referencia a sus portales y los portales a ella, de manera que pueda ser notificada cuando un portal se destruye. Lo anterior permitiría resolver escenarios tales como el disparo de un portal donde ya hay uno propio o ajeno, el reseteo de los portales, etcétera.

- Integración del procesamiento de eventos en la clase *Mundo*: Que debió refactorizarse ya que un hilo cliente debía interactuar vía eventos tanto con salas como con el mundo en sí, por lo que dicho comportamiento debió ser cambiado a último momento.

1.5. Herramientas

Herramientas auxiliares que utilizaron para hacer el proyecto, como editor de interfaz gráfica, control de versiones, herramienta para generar documentación, herramientas para debug, etcétera.

- Control de versiones: `git` mediante GitHub.
- Debugging: `GDB`.
- Persistencia de escenarios de juego: YAML (librería: `yaml-cpp`).
- Físicas del juego: `Box2D`.
- Herramienta para parte gráfica: `SDL2`.
- Otras librerías usadas: `sdl-image`, `sdl-mixer`.
- Desarrollo: `Sublime Text` y `Visual Studio Code`.

1.6. Conclusiones

El desarrollo del proyecto, al momento de la primera entrega, permite elaborar las siguientes conclusiones:

1.6.1. Servidor

Para el servidor, interesa destacar la dificultad del desarrollo, no tanto en la implementación, que fue bien cubierta durante las clases de la materia, sino en el tiempo requerido para idear una solución viable a cada problema que surgía, de modo que no implicara una reestructuración total del código y que resultara fácil de integrar con los demás módulos del servidor y el cliente.

1.6.2. Cliente

Como conclusion en lo que al cliente respecta, es importante darse cuenta de en que consiste cada ciclo del juego y lograr la abstraccion adecuada. Tambien hay que destacar que hay que tener bien presente como se efectuara la comunicacion con el servidor y como distribuir estas tareas en hilos diferentes al de dibujado.

1.6.3. Integración

En la integración del cliente y el servidor resultó muy importante el desarrollo de los Eventos que los mismos debían intercambiar: los mismos debieron estar muy desacoplados de los módulos restantes de modo que pudieran agregarse, quitarse o modificarse eventos sin la necesidad de modificaciones extensas. La especialización de eventos individuales permitió facilidades en su manejo y eficiencia al ser enviados durante la comunicación.

2. Documentación Técnica

2.1. Requerimientos de software

Para el compilado, desarrollo, prueba y depurado del proyecto es necesario: (Basado en lo utilizado, podría funcionar con versiones inferiores/diferentes de las herramientas)

- C++11
- Linux Ubuntu 18.04.2 LTS
- SDL: libsdl2-dev, libsdl-mixer-dev, libsdl-image-dev
- libyamlcpp-dev v0.5

2.2. Descripción general

Son dos aplicaciones, una para el servidor y otra para el cliente (jugador).

Al ser un juego en tiempo real, todos los clientes deben actualizar sus movimientos al servidor. Aquí se encuentra el modelo del juego, que se actualiza según los eventos recibidos por el cliente. Una vez procesados los eventos, el servidor envía el estado actual del juego a todos los clientes a un ritmo constante. De esta forma los clientes se mantienen actualizados entre sí y todos los cambios son controlados por el servidor. El servidor es una aplicación de consola.

El cliente es una aplicación gráfica hecha con SDL. Recibe el estado del juego actual enviado por el servidor y lo grafica con SDL. A su vez, obtiene las acciones del cliente (mouse, teclado, etc), las procesa en forma de eventos y se las envía al servidor para actualizar el modelo.

2.3. Módulos compartidos

2.3.1. Descripción General

Aqui se incluyen aquellas clases utilizadas tanto por el servidor como con el cliente.

2.3.2. Clases

- Cola: Cola no bloqueante. El metodo de extraccion devuelve un booleano segun si hay un objeto para extraer o no.

- ColaBloqueante: Cola protegida. Permite agregar y obtener elementos, mientras la cola esté activa.
- Conversor: Transforma coordenadas de servidor a coordenadas de cliente. Consiste en escalar y centrar.
- Evento: Modela un evento de juego. Cuenta con la capacidad de enviarse y recibirse por socket y de actualizar a quien lo maneje.
- Handler: Interfaz a ser implementada por quien maneje un evento.
- Serializador: Recibe el tipo de un evento por socket y segun el, crea un evento.
- Socket: Funcionalidades de los sockets tanto del cliente como del servidor.
- Thread: Encapsula un std::thread. Clase abstracta, a ser implementada por un hilo en el programa.

2.3.3. Descripción de archivos y protocolos

Protocolo:
Envio de eventos:

- Se envia el entero correspondiente al tipo del evento.
- Luego, se envian los parametros correspondientes al tipo del evento a enviar, todos enteros, sin un orden necesario.

Recepcion de eventos:

- Se recibe el entero correspondiente al tipo del evento.
- Luego, se reciben los parametros correspondientes al tipo del evento a enviar, todos enteros, en el orden en el que fueron enviados.

2.4. Módulo Cliente

2.4.1. Descripción General

A grandes rasgos, el cliente se ocupa de tres cosas: Actualizarse segun los cambios recibidos del servidor, dibujar la escena actual teniendo en cuenta dichas actualizaciones, y manejar los eventos de mouse y teclado, que se enviaran al servidor. Para la actualizacion, un hilo recibidor de eventos encola

los mismos, recibidos por socket desde el servidor, para ser enviados al hilo principal de dibujado. Esta cola es no bloqueante, ya que se deben poder seguir dibujando animaciones (ej. idle personaje, animacion del acido) aunque no suceda nada. El dibujado es, para cada objeto del juego, renderizarlo en su posicion inicial mas un delta correspondiente a la camara. Los eventos se manejan mediante la funcion PollEvents de SDL, y segun su tipo, se envia dicho evento al servidor. Para esto se crea el evento y se lo inserta en una cola bloqueante, de donde lo levanta un hilo enviador y lo envia por socket.

2.4.2. Clases

A continuacion las clases principales del cliente.

- SeleccionadorEscenas: Se encarga de ejecutar el ciclo del cliente mencionado arriba, eligiendo la escena correspondiente para cada momento.
- EscenaBase: Clase abstracta con las funciones del ciclo del cliente, a ser implementadas por cada escena concreta.
- EscenaJuego: Escena donde transcurre el juego, almacenara todos los objetos de juego que levanta del yaml y los procesara segun el ciclo del cliente.
- CreadorTexturas: Factory para objetos del juego. Se crea una vista de objeto segun los parametros especificados.
- VistaObjeto: Clase abstracta para una vista de objeto. Contiene una referencia a su textura y un vector de areas, usadas para las animaciones.
- RecibidorEventos: Hilo que recibe eventos del socket y los encola en una cola no bloqueante para comunicar con el hilo principal.
- EnviadorEventos: Hilo que envia eventos al socket, obtenidos de una cola bloqueante donde inserta eventos el hilo principal.
- Audio: Modela el audio de SDL, reproducira la musica de fondo y los efectos de sonido.
- SdlTexture: Levanta una textura y la renderiza. Cuenta con la capacidad de alterar su opacidad y su color.

2.4.3. Diagramas UML

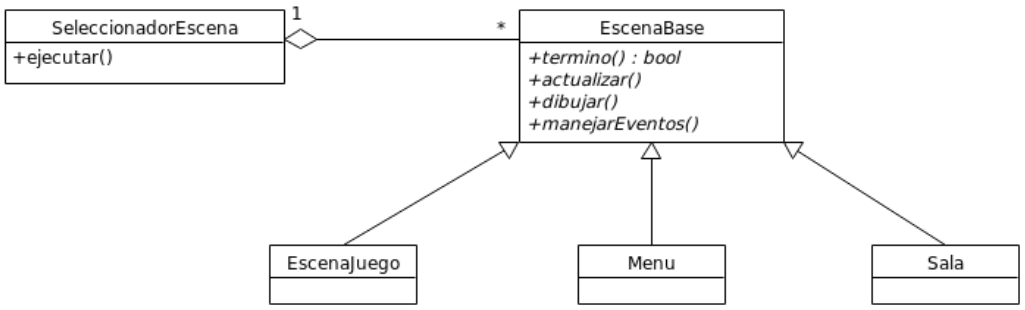


Figura 1: Diagrama de clases para las Escenas

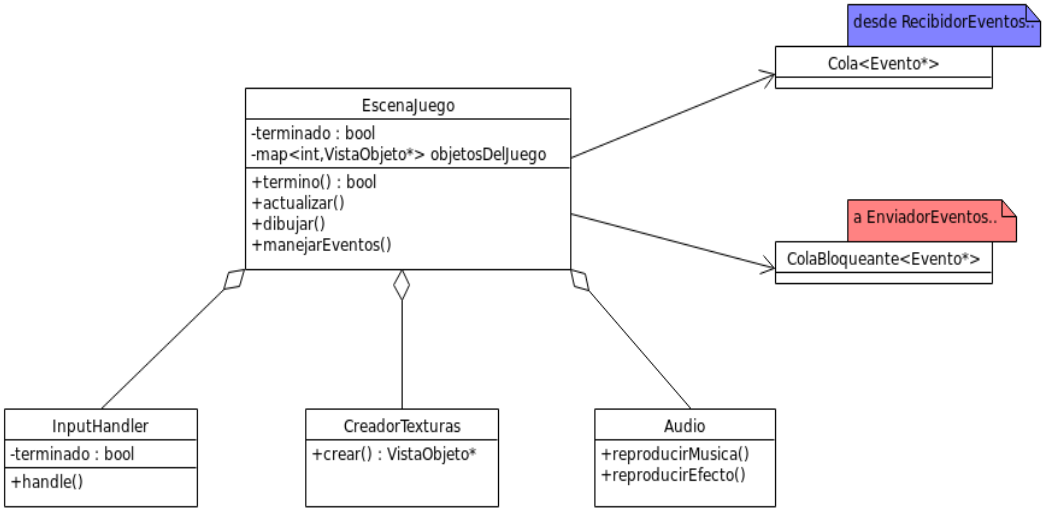


Figura 2: Diagrama de clases para el juego.

2.5. Módulo Servidor

2.5.1. Descripción General

A grandes rasgos, el módulo servidor consta de dos partes: una relacionada al juego en sí mismo y otra relacionada a la comunicación a través de la red. En la primera, se implementan las funcionalidades relacionadas a la simulación y eventos del juego, mientras que en la segunda abstrae la creación de diferentes salas y partidas, manejando las conexiones de los usuarios y recibiendo sus eventos.

2.5.2. Clases

Las clases más interesantes del módulo servidor se detallan a continuación:

- `Aceptador:` Hilo de ejecución que recibe conexiones de sockets entrantes. Para cada uno de los sockets recibidos, lanza un hilo EscuchadorCliente que será el encargado de recibir los eventos del mismo.
- `EscuchadorCliente:` Hilo encargado de recibir los eventos a través de la red. Posee un destinatario al cual reportará sus eventos para que sean manejados. El destinatario puede cambiar en tiempo de ejecución.
- `Servidor:` Engargado de lanzar el hilo aceptador, y responsable de crear *Salas*, que iniciarán partidas creando un *Mundo*.
- `Mundo:` Encargado de mantener la simulación del juego, entidades y sus posiciones. Maneja eventos recibidos de los jugadores.
- `Identificable:` clase implementada para brindar un identificador único a cada entidad relevante al juego. El mismo se utiliza para la consistencia entre el mundo y la representación en cada cliente.
- `Colisionable:` Clase de la cual heredan las entidades y superficies del juego, implementa la lógica de colisiones a través del patrón *Double Dispatch*.
- `Fisicas`: Capa de abstracción entre la lógica del juego Portal y la librería Box2D, utilizada para simular las físicas.
- `Transformación:` Clase abstracta que constituye un *functor*, se utiliza para encolar las transformaciones que deben aplicarse en el mundo. Se hizo necesaria ya que *Box2D* no permite modificar el mundo inmediatamente después de una colisión, y para el filtrado de colisiones evitando agregar/destruir una entidad repetidas veces.

2.5.3. Diagramas UML

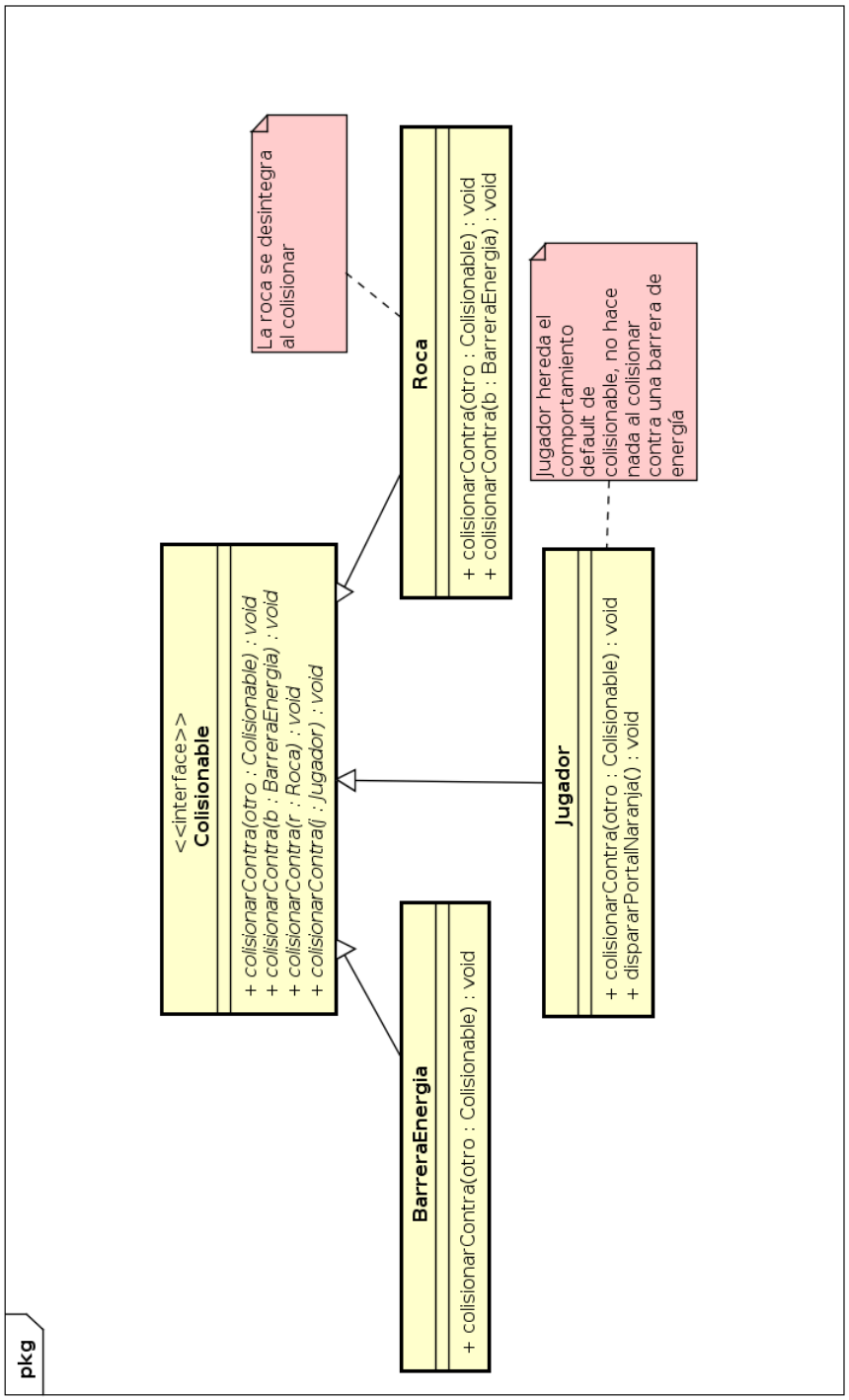


Figura 3: Manejo de colisiones

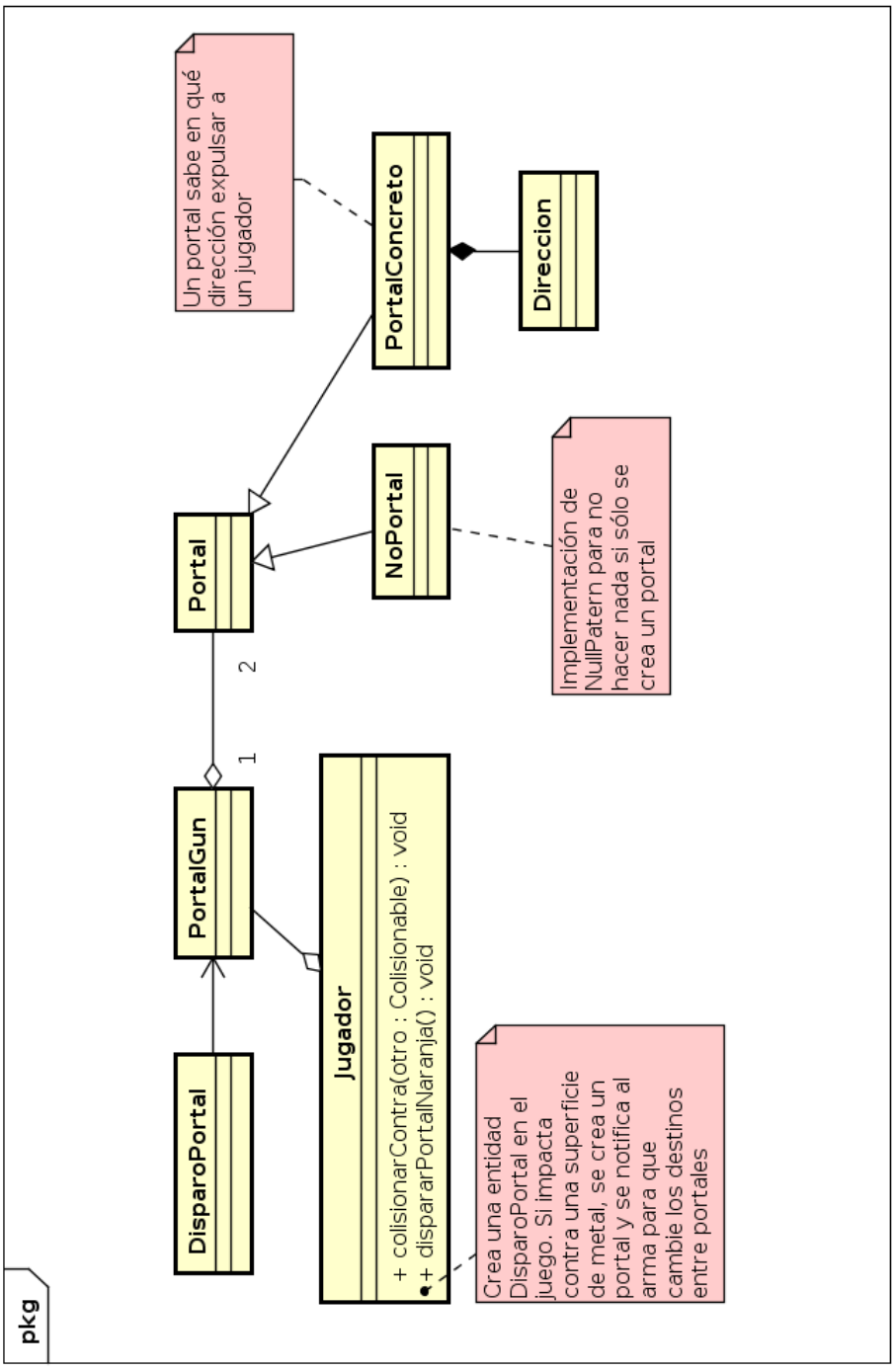


Figura 4: Diagrama de clases para el juego.

2.6. Programas intermedios y de prueba

Como programas intermedios y de prueba, se utilizó una implementación *dummy* del servidor, que devolvía todo evento recibido, y la utilidad *TestBed*, provista por `Box2D`.

2.7. Código fuente

El código fuente (tanto de la aplicación cliente como servidor) puede consultarse en el apendice B.

3. Manual de Usuario

Esta es una guía de instalación que permitirá obtener, compilar y ejecutar el juego *Portal*. Para su ejecución, el juego requiere que un servidor se esté ejecutando, y al menos un cliente para poder iniciar una partida. La instalación del cliente y el servidor, además de su ejecución, se detallan a continuación.

3.1. Instalación

3.1.1. Requerimientos de software

Indispensables:

- Aplicación `cmake`.
- Aplicación `make`.
- Librería `SDL2`.
- Librería `SDL2-image`.
- Librería `SDL2-mixer`.

Recomendados:

- Sistema Operativo: Ubuntu 18.04 LTS.
- Aplicación `git`.

3.1.2. Requerimientos de hardware

- 2 GB RAM.
- Placa de video capaz de soportar textiras de 4096x4096 píxeles.
- Conexión de red para partidas multijugador.
- Procesador Dual-Core 2.0GHz.

3.1.3. Proceso de instalación

A continuación se describe el proceso de instalación de las librerías. Luego, se muestra cómo instalar y ejecutar tanto el servidor como el cliente. Abriendo una terminal, se debe ejecutar:

Librerías de SDL2

```
sudo apt-get install libsdl2-dev libsdl2-image-dev
libsdl2-mixer-dev
```

Make

```
sudo apt-get install build-essential
```

CMake

```
sudo apt-get install cmake
```

git

```
sudo apt-get install git
```

Una vez instaladas y/o actualizadas las aplicaciones anteriores, se debe obtener el código fuente. Esto se puede realizar de dos maneras: mediante `git` o descargando los archivos fuente desde la página del repositorio que se encuentra en la portada. Para obtener el código mediante `git`, crear un directorio y ejecutar en una terminal:

Para obtener los fuentes

```
git clone https://github.com/sberoch/PortalTaller
```

Para instalar tanto el cliente como el servidor, ejecutar:

Instalación

```
mkdir build && cd build
cmake ..
make && make install
```

En la carpeta build resultarán dos ejecutables, `portal_server` para el servidor y `portal_cliente` para el cliente.

3.2. Forma de uso

Luego de realizar la instalación, desde la carpeta build ejecutar:

Ejecución del servidor

```
./portal_server puerto
```

y en una terminal nueva, en el mismo directorio:

Ejecución del cliente

```
./portal_cliente
```

Apéndices

A. Enunciado

A continuación se encuentra anexado el enunciado del trabajo práctico, que contiene una descripción detallada del software a implementar.

Descripción

El juego consiste en un escenario constituido por bloques, rocas y compuertas en el que los jugadores (representados por Chell) debe moverse y sortear obstáculos usando su ingenio y su *portal gun* o *the Aperture Science Handheld Portal Device*.

El objetivo de cada nivel o escenario es poder que **todos** los jugadores lleguen al pastel (un lugar específico del escenario).

La muerte de uno de los jugadores **no** implica que el resto no pueda seguir jugando: es válido que, si todos los jugadores menos uno pudieron llegar al pastel, el jugador restante se *eutanasie* (suicide) o que sus compañeros lo *eutanasien* (lo maten), para así ganar, todo por el bien común y de la ciencia.

En este sentido, aunque el espíritu del juego es *cooperativo*, puede tornarse en *competitivo*.

Bloques

El escenario estará compuesto por bloques.

Bloques de roca: no ofrecen una superficie lo suficientemente plana para la formación de un portal ni para la reflexión de las bolas de energía.

Bloques de metal: sus superficies planas permiten la creación de portales y la reflexión de las bolas de energía.

Bloques de metal en diagonal: al igual que los bloques de metal estos permiten la creación de portales y la reflexión de bolas de energía. La particularidad de este bloque es el ángulo 45 grados de una de sus superficies. Hay 4 variantes de este tipo de bloques, una por cada ángulo.

Los jugadores y otros objetos pueden caminar por sobre los bloques; en el caso de los bloques en diagonal, tanto los jugadores como los objetos se resbalan y caen debido a la pendiente de la superficie en diagonal.

Ácido

El ácido se encuentra depositado sobre uno o varios bloques y es letal para los jugadores.

Botones

Los botones se encuentran sobre los bloques y son activados por peso: mientras un jugador o alguna roca estén sobre él el botón quedará en estado activado; sin peso, el botón se desactiva.

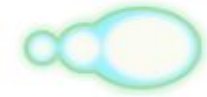
Rocas

Las rocas pueden ser cargadas y depositadas por los jugadores. Hay diferentes tipos de rocas pero todas ellas tienen la misma funcionalidad y la única diferencia es la imagen de cada roca.

Emisores y receptores de energía

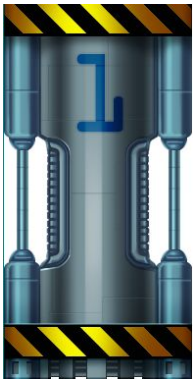
En el escenario pueden haber uno o más emisores de energía. Estos lanzan con cierta frecuencia *bolas de energía* que se mueven horizontal o verticalmente. Los receptores en cambio son elementos pasivos que esperan recibir una bola de energía y cuando lo hacen quedan activados.

Bolas de energía



Son emitidas por los *emisores*, sirven para activar a los *receptores* y se mueven a una velocidad determinada en dirección horizontal o vertical y rebotan (reflejan) sobre superficies de *metal* (sobre superficies en diagonal estas cambian de dirección). Luego de un cierto tiempo estas desaparecen.

Compuertas



Las compuertas no permiten el paso de ningún objeto y para abrirlas los jugadores deberán activar uno o varios botones o receptores de energía.

La combinación dependerá de cada compuerta: una puede abrirse con sólo mantener presionado un botón, otra puede activarse presionando dos botones **y (and)** un receptor, otra un botón **o (or)** otro botón.

Otras combinaciones son posibles incluyendo el operador **negación (not)**.

Barreras de energía

Las barreras de energía permiten el paso de los jugadores y las bolas de energía pero no las rocas ni de los rayos para crear los portales. Si un jugador traspasa la barrera cargando una roca esta se desintegra. Hay barreras horizontales como verticales.

Jugador

Los jugadores encarnaran a Chell, la protagonista del juego original Portal [1]. Chell es capaz de moverse lateralmente, de saltar y de disparar un rayo para la creación de portales con su *portal gun* o *the Aperture Science Handheld Portal Device*.
Chell cuenta con un equipamiento especial que evita que sufra daños al caer, no importa desde qué tan alto. Sin embargo ella es vulnerable y morirá si le cae una roca encima, se le cierra una compuerta encima, cae en ácido o es impactada por una bola de energía.

Creación de portales

Hay 2 portales para cada jugador, el azul y el naranja. Una vez creados estos forman un túnel bidireccional que puede teletransportar a los jugadores, rocas y bolas de energía instantáneamente.

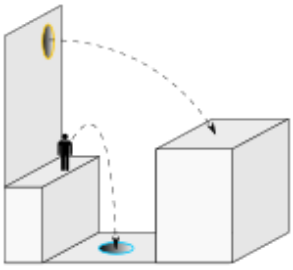
La orientación del portal dependerá de la orientación de la superficie del bloque en donde se creó: horizontal, vertical o diagonal.

Cada jugador podrá crear tantos portales como quiera pero la creación de un portal azul implica que el portal azul anterior del mismo jugador se desactiva (desaparece). Lo mismo aplica al portal naranja.

El jugador puede usar el mouse para indicar la dirección hacia a donde disparar un rayo en línea recta que al impactar crea un portal. Puede crear un portal azul o uno naranja haciendo click izquierdo o derecho.

El o los portales de un jugador pueden desactivarse a modo de *reset* usando el teclado.

Conservación del movimiento relativa



Cuando un jugador u otro elemento ingresa a un portal a cierta velocidad este sale por el portal opuesto con la misma velocidad en magnitud y misma dirección **respecto** a la normal del portal.

Desde la óptica del jugador o elemento que se mueve entre portales su *cantidad de movimiento* es *conservada* tanto en magnitud como en dirección.

Cámara

La cámara muestra una porción del escenario (los escenarios pueden ser muy largos y no entrar en la vista de la cámara) y debe enfocarse en el jugador y seguirlo a medida que se desplaza.

Animaciones

El juego no debe mostrar imágenes estáticas sino pequeñas animaciones para darle mayor realismo [4]:

- El movimiento de los jugadores: cuando se desplazan, saltan, disparan, mueren.
- El movimiento de las bolas de energía, las compuertas y los portales.
- Activación y desactivación de botones y otros elementos..

Sonidos

Como todo juego se debe reproducir sonidos para darle realismo a los eventos y acciones que suceden[5]:

- Cuando hay disparos.
- Cuando un jugador salta.
- Cuando un jugador muere.

Si la cantidad de eventos que suceden es muy grande, algunos sonidos pueden ser evitados para no saturar al jugador con tanta información.

Musica ambiente

El juego debe reproducir una música ambiente, con un volumen relativamente bajo[5].

Interfaz del jugador

El juego debe poder renderizarse en full screen y en modo ventana con el tamaño de esta configurable.

Cada jugador tendrá un color asociado de tal forma que se puedan distinguir las distintas Chells.

Pin tool

El jugador puede crear un *pin*, una marca de donde uno de sus compañeros debe crear un portal. La mecánica de la creación de pins es igual a la de creación de portales (usando el mouse) pero presionando otro botón/tecla para diferenciar la creación de uno de otro.
Los pins desaparecen luego de cierto tiempo o cuando otro pin del mismo jugador es creado.

Aplicaciones Requeridas

Cliente

Se deberá implementar un cliente gráfico para que el usuario pueda conectarse al servidor, crear o unirse a una partida eligiendo el escenario a jugar.
Con la aplicación cliente el jugador podrá además de jugar iniciar o frenar la grabación en video de la partida actual.

Servidor

Se deberá implementar un servidor con soporte de múltiples partidas en simultáneo. Deberá poder indicarle a los clientes que se conecta qué escenarios hay disponibles así como también que partidas ya están creadas y están disponibles para que el usuario pueda unirse a alguna de ellas.

Todos los atributos del juego (velocidad, altura de salto, etc) deben ser configurables por archivo.

> *Es importante que todos los parámetros sean configurables: permite que que se ajusten para tener un juego más balanceado y divertido a la vez que le permite a los docentes realizar pruebas.*

Editor

Se deberá implementar un editor de escenarios que permita:

- Crear nuevos escenarios (o niveles) o editar previos.
- Definir la locación de los bloques, rocas, compuertas y otros elementos que forman el escenario.
- Definir la locación inicial de los jugadores y cuantos jugadores son requeridos para que se unan y puedan jugar a dicho escenario y la locación del pastel.
- Cada nivel debe tener además una imagen estática que sirva de fondo.
- Definir las reglas (lógica booleana) para abrir o cerrar cada compuerta en función de la activación o desactivación de uno o varios botones y/o receptores de energía. *Nota: hacer que cada botón y receptor de energía tenga un nombre.*
- Verificar ciertas condiciones mínimas como al menos una Chell y un solo pastel.

Distribución de Tareas Propuesta

Con el objetivo de organizar el desarrollo de las tareas y distribuir la carga de trabajo, es necesario planificar las actividades y sus responsables durante la ejecución del proyecto. La siguiente tabla plantea una posible división de tareas de alto nivel que puede ser tomada como punto de partida para la planificación final del trabajo:

	Alumno 1 **Servidor - Modelo**	**Alumno 2** **Cliente - Modelo**	**Alumno 3** **Cliente - Editor**
Semana 1 (30/04/2019)	- Draft del modelo (incluyendo lógica del juego y partidas multijugador) - Prueba de concepto con Box2D.	- Mostrar una imagen. - Mostrar una animación. - Mostrar ambas en un lugar fijo o desplazándose por la pantalla (movimiento).	- Draft del cliente y del editor (*wireframe*). - Prueba de concepto con ffmpeg.
Semana 2 (07/05/2019)	- Escenario (bloques, rocas y otras cosas no dinámicas)	- Renderizado del escenario incluyendo la cámara.	- Edición de un escenario básico con solo objetos estáticos.
Semana 3 (14/05/2019)	- Chell (jugador), carga/descarga de rocas y bolas de energía y otros elementos dinámicos.	- Animación de los elementos dinámicos.	- Edición de un escenario incluyendo la locación de las Chells y del pastel.
Semana 4 (21/05/2019)	- Portales y física completa. Lógica de compuertas	- Finalización de la parte gráfica incluyendo la pin tool.	- Edición de las condiciones booleanas para cada compuerta.
Semana 5 (28/05/2019)	- Servidor multipartidas con partidas multijugador. Condiciones de victoria y derrota.	- Música y sonido. Pantallas de login, creación de partidas y de unirse a partidas.	- Captura de video de una partida: inicio y frenado a voluntad.
Semana 6 (04/06/2019)	- Testing - Correcciones y *tuning* del Servidor - Documentación	- Testing - Correcciones y *tuning* del Cliente - Documentación	- Testing - Correcciones y *tuning* del Editor - Documentación
Entrega el 11/06/2019			
Semana 7 (11/06/2019)	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación
Semana 8 (18/06/2019)	- Testing - Correcciones sobre la primer entrega - Armado del entregable	- Testing - Correcciones sobre primer entrega - Armado del entregable	- Testing - Correcciones sobre primer entrega - Armado del entregable
Reentrega el 25/06/2019			

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema se debe realizar en C++11 utilizando librerías *gtkmm, SDL y/o qt*.

2. Los archivos de configuración deben ser almacenados en formato *YAML* [6]. A tal fin, y con el objetivo de minimizar tiempos y posibles errores, se permiten distintas librerías externas (consultar sitio de la cátedra). No está permitido utilizar una implementación propia de lectura y escritura de *YAML*.
3. Para la simulación de la física del juego se debe usar el framework *Box2D* [2].
4. Para la grabación del video de las partidas se debe usar *ffmpeg* [3].
5. Es condición necesaria para la aprobación del trabajo práctico la entrega de la documentación mínima exigida (consultar sitio de la cátedra). Es importante recordar que cualquier elemento faltante o de dudosa calidad pone en riesgo la aprobación del ejercicio.
6. Entrega de uno o varios escenarios con la suficiente diversidad de elementos a tal fin que sea fácil mostrar las funcionalidades implementadas.
7. De forma opcional, se sugiere la utilización de alguna librería del estilo xUnit [7]. Si bien existen varias librerías disponibles en lenguaje C++ [8], se recomienda optar por CxxTest [9] o CppUnit [10].

Referencias

[1] Portal: https://en.wikipedia.org/wiki/Portal_(video_game)
[2] Box2D: http://box2d.org/manual.pdf
[3] ffmpeg: https://ffmpeg.org/
[4] Sprites: https://www.spriters-resource.com/pc_computer/mightyswitchforcehyperdriveedition/
[5] Efectos y música ambiente: https://www.youtube.com/watch?v=2pByCegIjpU
https://www.youtube.com/watch?v=Kgny67NTtw0 https://theportalwiki.com/wiki/GLaDOS_voice_lines
[6] YAML: https://es.wikipedia.org/wiki/YAML
[7] Frameworks XUnit: http://en.wikipedia.org/wiki/XUnit
[8] Variantes XUnit para C/C++: http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C.2B.2B
[9] CxxTest: http://cxxtest.com/
[10] CppUnit: http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main_Page

B. Código Fuente

jun 11, 19 16:12 **server_main.cpp** Page 1/1

```cpp
#include "server.h"
#include "iostream"

int main(int argc, char const *argv[]) {
    Servidor s(argv[1]);
    try {
        s.correr();
    } catch(const std::exception& e) {
        std::cout << e.what() << '\n';
    }
    return 0;
}
```

jun 11, 19 16:12 **server.h** Page 1/1

```cpp
#ifndef __SERVER_H__
#define __SERVER_H__

#include <string>

#include "../Common/Socket.h"
#include "../Common/handler.h"

#define HOST "localhost"
#define CONDICION_SALIR 'q'

class Servidor : public Handler {
    private:
    Socket sktAceptador_;

    public:
    Servidor(const std::string& unPuerto);
    void correr();
    virtual void manejar(Evento& evento) override;
};

#endif
```

jun 11, 19 16:12 **server.cpp** Page 1/1

```cpp
#include "server.h"

#include <iostream>

#include "red/aceptador.h"

#include "../Common/value_protected.h"
#include "server_config.h"


Servidor::Servidor(const std::string& unPuerto) {
    sktAceptador_.vincularYEscuchar(unPuerto.c_str(), CONFIG.MAX_EN_ESPERA);
}

void Servidor::correr() {
    ValueProtected<bool> seguirCorriendo(true);
    Aceptador aceptador(sktAceptador_, seguirCorriendo, *this);
    aceptador.iniciar();
    char c;
    while ((c = std::cin.get()) ≠ CONDICION_SALIR) {
        // pass
    }
    seguirCorriendo.set(false);
    sktAceptador_.shutdown();
    aceptador.cerrar();
}

void Servidor::manejar(Evento& unEvento) {

}
```

jun 11, 19 16:12 **server_config.h** Page 1/2

```cpp
#ifndef __SERVER_CONFIG_H__
#define __SERVER_CONFIG_H__

struct Config {
    /* CONSTANTES RELACIONADAS A LA RED */
    int MAX_EN_ESPERA = 20;
    /* FIN CONSTANTES RELACIONADAS A LA RED */

    /* CONSTANTES DE SIMULACION*/
    float GRAVEDAD_X = 0.0f;
    float GRAVEDAD_Y = -10.0f;
    float TIME_STEP = 1.0f / 30.0f;
  int VELOCITY_ITERATIONS = 15;
  int POSITION_ITERATIONS = 10;
    /* FIN CONSTANTES DE SIMULACION*/

    /* TAMAÃM-^QOS */
    /*
    /* Tener en cuenta que todos los tamaÃ±os definidos son
    /* refiriÃ©ndose a las hitboxes, por lo que estÃ¡n pensados
    /* para mejorar la experiencia de juego
    */
    float SIZE_SENSOR_METAL_CUADRADO_X = 0.3f;
    float SIZE_SENSOR_METAL_CUADRADO_Y = 0.2f;

    float SIZE_SENSOR_METAL_DIAGONAL_X = SIZE_SENSOR_METAL_CUADRADO_X * 1.4142f;
    float SIZE_SENSOR_METAL_DIAGONAL_Y = SIZE_SENSOR_METAL_CUADRADO_Y;

    float SIZE_PORTAL_X = 0.05f;
    float SIZE_PORTAL_Y = 0.25f;

    float SIZE_BLOQUE_X = 1.0f;
    float SIZE_BLOQUE_Y = 1.0f;

    float SIZE_BOTON_X = 0.8f;
    float SIZE_BOTON_Y = 0.1f;

    float SIZE_JUGADOR_X = 0.25f;
    float SIZE_JUGADOR_Y = 0.375f;

    float SIZE_RADIO_BOLA_ENERGIA = 0.5f;

    float SIZE_RADIO_DISPARO_PIN = 0.025f;

    float SIZE_RADIO_DISPARO_PORTAL = 0.1f;

    float SIZE_RADIO_ROCA = 0.35f;
    /* FIN TAMAÃM-^QOS */

    /* DENSIDADES */
    float DENSIDAD_DEFAULT = 1.0f;
    float DENSIDAD_BLOQUE = DENSIDAD_DEFAULT;
    float DENSIDAD_JUGADOR = DENSIDAD_DEFAULT;
    float DENSIDAD_BOLA_ENERGIA = DENSIDAD_DEFAULT;
    float DENSIDAD_DISPARO_PIN = DENSIDAD_DEFAULT;
    float DENSIDAD_DISPARO_PORTAL = DENSIDAD_DEFAULT;
    float DENSIDAD_ROCA = DENSIDAD_DEFAULT;
    /* FIN DENSIDADES*/

    /* ROZAMIENTOS */
    float ROZAMIENTO_SUPERFICIE = 0.0f;
    /* FIN ROZAMIENTOS */

};

extern Config CONFIG;
```

jun 11, 19 16:12 **server_config.h** Page 2/2

```
67
68  #endif
```

jun 11, 19 16:12 **server_config.cpp** Page 1/1

```
1  #include "server_config.h"
2
3  Config CONFIG;
```

jun 11, 19 16:12 **escuchador_cliente.h** Page 1/1

```
1  #ifndef __ESCUCHADOR_CLIENTE_H__
2  #define __ESCUCHADOR_CLIENTE_H__
3
4  #include "../../Common/Thread.h"
5  #include "../../Common/Socket.h"
6
7  // Forward declaration
8  class Handler;
9  class Socket;
10
11 class EscuchadorCliente : public Thread {
12     private:
13     Socket sktCliente_;
14     Handler* destinatario_;
15     bool finalizado_;
16
17     public:
18     EscuchadorCliente(Socket∧ skt, Handler* unDestinatario);
19     virtual void ejecutar() override;
20     void stop();
21     void cambiarDestinatario(Handler* nuevoDestinatario);
22     bool finalizado();
23 };
24
25 #endif
```

jun 11, 19 16:12 **escuchador_cliente.cpp** Page 1/1

```
1  #include "escuchador_cliente.h"
2
3  #include "../../Common/handler.h"
4  #include "../../Common/Evento.h"
5
6  EscuchadorCliente::EscuchadorCliente(Socket∧ skt, Handler* unDestinatario) :
7      destinatario_(unDestinatario) {
8      sktCliente_ = std::move(skt);
9      finalizado_ = false;
10 }
11
12 void EscuchadorCliente::cambiarDestinatario(Handler* nuevoDestinatario) {
13     destinatario_ = nuevoDestinatario;
14 }
15
16 bool EscuchadorCliente::finalizado() {
17     return finalizado_;
18 }
19
20 void EscuchadorCliente::ejecutar() {
21     int idCrearJugador = 214;
22     int idJugador = 29;
23     sktCliente_.enviarInt(idCrearJugador);
24     sktCliente_.enviarInt(idJugador);
25 }
26
27 void EscuchadorCliente::stop() {
28     sktCliente_.shutdown();
29 }
```

jun 11, 19 16:12 **aceptador.h** Page 1/1

```h
#ifndef __ACEPTADOR_H__
#define __ACEPTADOR_H__

#include <vector>

#include "../../Common/Thread.h"
#include "../../Common/value_protected.h"

// Forward declaration
class Socket;
class Servidor;
class EscuchadorCliente;

class Aceptador : public Thread {
    private:
    Socket& skt_;
    ValueProtected<bool>& seguirCorriendo_;
    Servidor& servidor_;
    std::vector<EscuchadorCliente*> clientes_;

    public:
    Aceptador(Socket& skt, ValueProtected<bool>& seguirCorriendo, Servidor& servidor);
    virtual void ejecutar() override;
    virtual void cerrar() override;
};

#endif
```

jun 11, 19 16:12 **aceptador.cpp** Page 1/1

```cpp
#include "aceptador.h"

#include "../server_config.h"

#include "../../Common/Socket.h"
#include "escuchador_cliente.h"
#include "../server.h"


Aceptador::Aceptador(Socket& skt, ValueProtected<bool>& seguirCorriendo, Servidor& servidor) :
    skt_(skt),
    seguirCorriendo_(seguirCorriendo),
    servidor_(servidor) {
}

void Aceptador::ejecutar() {

    while (seguirCorriendo_()) {
        try {
            Socket aceptado = skt_.aceptar();
            clientes_.push_back(new EscuchadorCliente(std::move(aceptado), &servidor_));
            clientes_.back()→iniciar();
            auto it = std::begin(clientes_);
            while (it ≠ std::end(clientes_)) {
                if ((*it)→finalizado()) {
                    (*it)→cerrar();
                    delete (*it);
                    it = clientes_.erase(it);
                } else {
                    ++it;
                }
            }
        }
        catch(const std::exception& e) {
            continue;
        }
    }
}

void Aceptador::cerrar() {
    auto it = std::begin(clientes_);
    while (it ≠ std::end(clientes_)) {
        (*it)→stop();
        (*it)→cerrar();
        delete (*it);
        ++it;
    }
    Thread::cerrar();
}
```

jun 11, 19 16:12 **superficie_metal.h** Page 1/1

```
1  #ifndef __SUPERFICIE_METAL_H__
2  #define __SUPERFICIE_METAL_H__
3
4  #include "superficie.h"
5  #include "../fisicas/movimiento/direccion.h"
6
7  // Forward declaration
8  class Fisicas;
9  class Direccion;
10
11 class SuperficieMetal : public Superficie {
12     private:
13     Direccion direccion_;
14
15     public:
16     SuperficieMetal(Fisicas& unasFisicas, Direccion& unaDireccion);
17     virtual ~SuperficieMetal();
18     virtual void colisionarContra(Colisionable& unColisionable) override;
19 };
20
21 #endif
```

jun 11, 19 16:12 **superficie_metal.cpp** Page 1/1

```
1  #include "superficie_metal.h"
2
3  SuperficieMetal::SuperficieMetal(Fisicas& unasFisicas, Direccion& unaDireccion)
   :
4      Superficie(unasFisicas),
5      direccion_(unaDireccion.copiar()) {
6  }
7
8  SuperficieMetal::~SuperficieMetal() {
9  }
10
11 void SuperficieMetal::colisionarContra(Colisionable& unColisionable) {
12     unColisionable.colisionarContra(*this);
13 }
```

jun 11, 19 16:12 **superficie.h** Page 1/1

```
1  #ifndef __SUPERFICIE_H__
2  #define __SUPERFICIE_H__
3
4  #include "../colisionable.h"
5
6  class Superficie : public Colisionable {
7      public:
8      Superficie(Fisicas& unasFisicas);
9      virtual ~Superficie();
10 };
11
12 #endif
```

jun 11, 19 16:12 **superficie.cpp** Page 1/1

```
1  #include "superficie.h"
2
3  Superficie::Superficie(Fisicas& unasFisicas) :
4      Colisionable(unasFisicas) {
5  }
6
7  Superficie::~Superficie() {
8  }
```

jun 11, 19 16:12 **bloque.h** Page 1/1

```h
#ifndef __BLOQUE_H__
#define __BLOQUE_H__

#include "../colisionable.h"

class Bloque : public Colisionable {
    public:
    Bloque(Fisicas& unasFisicas);
    virtual ~Bloque();
    virtual void colisionarContra(Colisionable& otro) override;
};

#endif
```

jun 11, 19 16:12 **bloque.cpp** Page 1/1

```cpp
#include "bloque.h"

Bloque::Bloque(Fisicas& unasFisicas) :
    Colisionable(unasFisicas) {
}

Bloque::~Bloque() {
}

void Bloque::colisionarContra(Colisionable& otro) {
    otro.colisionarContra(*this);
}
```

jun 11, 19 16:12 **mundo.h** Page 1/1

```
#ifndef __MUNDO_H__
#define __MUNDO_H__

#include <map>
#include <queue>

#include "fisicas/fisicas.h"

// Forward declaration
class Colisionable;
class Posicion;
class Velocidad;
class Rotacion;
class Evento;

class Mundo {
    private:
    Fisicas fisicas_;
    std::map<int, std::shared_ptr<Colisionable>> bloques_;
    std::map<int, std::shared_ptr<Entidad>> entidades_;
    std::queue<std::shared_ptr<Evento>> eventos_;

    public:
    void agregarBloqueMetalCuadrado(Posicion& posicion, Rotacion& r);
    void agregarBloqueMetalTriangular(Posicion& posicion, Rotacion& r);
    void agregarJugador(Posicion& posicion);
    void moverJugador(int uuidDelJugador, Velocidad& v);
    void step();
    // Testing
    Fisicas* getFisicas() {return &fisicas_;}
};

#endif
```

jun 11, 19 16:12 **mundo.cpp** Page 1/3

```
#include "mundo.h"

#include "../server_config.h"

#include "superficies/bloque.h"
#include "superficies/superficie_metal.h"

#include "entidades/jugador.h"

#include "fisicas/formas/forma.h"
#include "fisicas/movimiento/direccion.h"
#include "fisicas/movimiento/posicion.h"
#include "fisicas/movimiento/rotacion.h"

#define NO_ROTADO 0.0f

void Mundo::agregarBloqueMetalCuadrado(Posicion& posicion, Rotacion& r) {
    // Se agrega un bloque de colision
    std::shared_ptr<Bloque> bloque(new Bloque(fisicas_));
    Forma forma(CONFIG.SIZE_BLOQUE_X, CONFIG.SIZE_BLOQUE_Y);
    bloques_[bloque→uuid()] = bloque;
    fisicas_.agregarBloqueRectangular(*bloque, posicion, forma);

    // Se agregan sensores de metal, que responden a colisiones
    Forma formaHorizontal(CONFIG.SIZE_SENSOR_METAL_CUADRADO_X,
        CONFIG.SIZE_SENSOR_METAL_CUADRADO_Y);
    Forma formaVertical(CONFIG.SIZE_SENSOR_METAL_CUADRADO_Y,
        CONFIG.SIZE_SENSOR_METAL_CUADRADO_X);

    Posicion posicionArriba = posicion + Posicion(0.0f, CONFIG.SIZE_BLOQUE_Y);
    Direccion direccionArriba(0.0f, 1.0f);
    std::shared_ptr<SuperficieMetal> metalArriba(new SuperficieMetal(fisicas_, d
ireccionArriba));
    bloques_[metalArriba→uuid()] = metalArriba;
    fisicas_.agregarSuperficie(*metalArriba, posicionArriba, formaHorizontal, r)
;

    Posicion posicionAbajo = posicion + Posicion(0.0f, (-1) * CONFIG.SIZE_BLOQUE
_Y);
    Direccion direccionAbajo(0.0f, -1.0f);
    std::shared_ptr<SuperficieMetal> metalAbajo(new SuperficieMetal(fisicas_, di
reccionAbajo));
    bloques_[metalAbajo→uuid()] = metalAbajo;
    fisicas_.agregarSuperficie(*metalAbajo, posicionAbajo, formaHorizontal, r);

    Posicion posicionIzquierda = posicion + Posicion((-1) * CONFIG.SIZE_BLOQUE_X
, 0.0f);
    Direccion direccionIzquierda(-1.0f, 0.0f);
    std::shared_ptr<SuperficieMetal> metalIzquierda(new SuperficieMetal(fisicas_
, direccionIzquierda));
    bloques_[metalIzquierda→uuid()] = metalIzquierda;
    fisicas_.agregarSuperficie(*metalIzquierda, posicionIzquierda, formaVertical
, r);

    Posicion posicionDerecha = posicion + Posicion(CONFIG.SIZE_BLOQUE_X, 0.0f);
    Direccion direccionDerecha(1.0f, 0.0f);
    std::shared_ptr<SuperficieMetal> metalDerecha(new SuperficieMetal(fisicas_,
direccionDerecha));
    bloques_[metalDerecha→uuid()] = metalDerecha;
    fisicas_.agregarSuperficie(*metalDerecha, posicionDerecha, formaVertical, r)
;
}

void Mundo::agregarBloqueMetalTriangular(Posicion& posicion, Rotacion& r) {
    // Se agrega un bloque de colision
    std::shared_ptr<Bloque> bloque(new Bloque(fisicas_));
```

```
58      Forma forma(CONFIG.SIZE_BLOQUE_X, CONFIG.SIZE_BLOQUE_Y);
59      bloques_[bloque→uuid()] = bloque;
60      fisicas_.agregarBloqueTriangular(*bloque, posicion, forma, r);
61
62      Forma formaHorizontal(CONFIG.SIZE_SENSOR_METAL_CUADRADO_X,
63          CONFIG.SIZE_SENSOR_METAL_CUADRADO_Y);
64      Forma formaVertical(CONFIG.SIZE_SENSOR_METAL_CUADRADO_Y,
65          CONFIG.SIZE_SENSOR_METAL_CUADRADO_X);
66      Forma formaDiagonal(CONFIG.SIZE_SENSOR_METAL_DIAGONAL_X,
67          CONFIG.SIZE_SENSOR_METAL_DIAGONAL_Y);
68
69      Posicion posicionArriba = posicion + Posicion(0.0f, 0.0f + CONFIG.SIZE_BLOQU
    E_Y);
70      Posicion posicionAbajo = posicion + Posicion(0.0f, -1*CONFIG.SIZE_BLOQUE_Y);
71      Posicion posicionIzquierda = posicion + Posicion(-1*CONFIG.SIZE_BLOQUE_Y, 0.
    0f);
72      Posicion posicionDerecha = posicion + Posicion(CONFIG.SIZE_BLOQUE_Y, 0.0f);
73
74      Direccion direccionArriba(0.0f, 1.0f);
75      Direccion direccionAbajo(0.0f, -1.0f);
76      Direccion direccionIzquierda(-1.0f, 0.0f);
77      Direccion direccionDerecha(-1.0f, 0.0f);
78
79      Rotacion no(0);
80
81      if (r.anguloGrados() ≡ 0) {
82          std::shared_ptr<SuperficieMetal> metalAbajo(new SuperficieMetal(fisicas_
    , direccionAbajo));
83          bloques_[metalAbajo→uuid()] = metalAbajo;
84          fisicas_.agregarSuperficie(*metalAbajo, posicionAbajo, formaHorizontal,
    no);
85
86          std::shared_ptr<SuperficieMetal> metalIzquierda(new SuperficieMetal(fisi
    cas_, direccionIzquierda));
87          bloques_[metalIzquierda→uuid()] = metalIzquierda;
88          fisicas_.agregarSuperficie(*metalIzquierda, posicionIzquierda, formaVert
    ical, no);
89
90          std::shared_ptr<SuperficieMetal> metalDiagonal(new SuperficieMetal(fisic
    as_, direccionArriba));
91          bloques_[metalDiagonal→uuid()] = metalDiagonal;
92          Rotacion delta(r.anguloGrados() + 135);
93          fisicas_.agregarSuperficie(*metalDiagonal, posicion, formaDiagonal, delt
    a);
94
95      } else if (r.anguloGrados() ≡ 90) {
96          std::shared_ptr<SuperficieMetal> metalAbajo(new SuperficieMetal(fisicas_
    , direccionAbajo));
97          bloques_[metalAbajo→uuid()] = metalAbajo;
98          fisicas_.agregarSuperficie(*metalAbajo, posicionAbajo, formaHorizontal,
    no);
99
100         std::shared_ptr<SuperficieMetal> metalDerecha(new SuperficieMetal(fisica
    s_, direccionDerecha));
101         bloques_[metalDerecha→uuid()] = metalDerecha;
102         fisicas_.agregarSuperficie(*metalDerecha, posicionDerecha, formaVertical
    , no);
103
104         std::shared_ptr<SuperficieMetal> metalDiagonal(new SuperficieMetal(fisic
    as_, direccionArriba));
105         bloques_[metalDiagonal→uuid()] = metalDiagonal;
106         Rotacion delta(r.anguloGrados() + 135);
107         fisicas_.agregarSuperficie(*metalDiagonal, posicion, formaDiagonal, delt
    a);
108
109     } else if (r.anguloGrados() ≡ 180) {
```



```
110         std::shared_ptr<SuperficieMetal> metalArriba(new SuperficieMetal(fisicas
    _, direccionArriba));
111         bloques_[metalArriba→uuid()] = metalArriba;
112         fisicas_.agregarSuperficie(*metalArriba, posicionArriba, formaHorizontal
    , no);
113
114         std::shared_ptr<SuperficieMetal> metalDerecha(new SuperficieMetal(fisica
    s_, direccionDerecha));
115         bloques_[metalDerecha→uuid()] = metalDerecha;
116         fisicas_.agregarSuperficie(*metalDerecha, posicionDerecha, formaVertical
    , no);
117
118         std::shared_ptr<SuperficieMetal> metalDiagonal(new SuperficieMetal(fisic
    as_, direccionAbajo));
119         bloques_[metalDiagonal→uuid()] = metalDiagonal;
120         Rotacion delta(r.anguloGrados() + 135);
121         fisicas_.agregarSuperficie(*metalDiagonal, posicion, formaDiagonal, delt
    a);
122
123     } else if (r.anguloGrados() ≡ 270) {
124         std::shared_ptr<SuperficieMetal> metalArriba(new SuperficieMetal(fisicas
    _, direccionArriba));
125         bloques_[metalArriba→uuid()] = metalArriba;
126         fisicas_.agregarSuperficie(*metalArriba, posicionArriba, formaHorizontal
    , no);
127
128         std::shared_ptr<SuperficieMetal> metalIzquierda(new SuperficieMetal(fisi
    cas_, direccionIzquierda));
129         bloques_[metalIzquierda→uuid()] = metalIzquierda;
130         fisicas_.agregarSuperficie(*metalIzquierda, posicionIzquierda, formaVert
    ical, no);
131
132         std::shared_ptr<SuperficieMetal> metalDiagonal(new SuperficieMetal(fisic
    as_, direccionAbajo));
133         bloques_[metalDiagonal→uuid()] = metalDiagonal;
134         Rotacion delta(r.anguloGrados() + 135);
135         fisicas_.agregarSuperficie(*metalDiagonal, posicion, formaDiagonal, delt
    a);
136     }
137 }
138
139 void Mundo::agregarJugador(Posicion& posicion) {
140     std::shared_ptr<Jugador> jugador(new Jugador(fisicas_));
141     Forma formaJugador(CONFIG.SIZE_JUGADOR_X, CONFIG.SIZE_JUGADOR_Y);
142     entidades_[jugador→uuid()] = jugador;
143     fisicas_.agregarEntidad(*jugador, posicion, formaJugador);
144 }
145
146 void Mundo::moverJugador(int uuidJugador, Velocidad& v) {
147     fisicas_.cambiarVelocidad(*entidades_[uuidJugador], v);
148 }
149
150 void Mundo::step() {
151     fisicas_.step();
152 }
```

jun 11, 19 16:12 **identificable.h** Page 1/1

```
1  #ifndef __IDENTIFICABLE_H__
2  #define __IDENTIFICABLE_H__
3
4  #include <mutex>
5
6  class Identificable {
7  private:
8      std::mutex mutex_;
9      const int UUID_;
10     static int contadorUUID_;
11     int proximoUUID();
12
13 public:
14     Identificable();
15     virtual ~Identificable();
16     int uuid();
17 };
18
19 #endif
```

jun 11, 19 16:12 **identificable.cpp** Page 1/1

```
1  #include "identificable.h"
2
3  int Identificable::contadorUUID_ = 0;
4
5  int Identificable::proximoUUID() {
6      std::lock_guard<std::mutex> lck(mutex_);
7      return ++contadorUUID_;
8  }
9
10 Identificable::Identificable() : UUID_(proximoUUID()) {
11 }
12
13 Identificable::~Identificable() {
14 }
15
16 int Identificable::uuid() {
17     return UUID_;
18 }
```

jun 11, 19 16:12 **transformacion.h** Page 1/1

```
1  #ifndef __TRANSFORMACION_H__
2  #define __TRANSFORMACION_H__
3
4  // Forward declaration
5  class Fisicas;
6
7  class Transformacion {
8      protected:
9      Fisicas& fisicas_;
10
11     public:
12     Transformacion(Fisicas& unasFisicas);
13     virtual ~Transformacion();
14     virtual void aplicar() = 0;
15 };
16
17 #endif
```

jun 11, 19 16:12 **transformacion.cpp** Page 1/1

```
1  #include "transformacion.h"
2
3  Transformacion::Transformacion(Fisicas& unasFisicas) :
4      fisicas_(unasFisicas) {
5  }
6
7  Transformacion::~Transformacion() {
8  }
```

jun 11, 19 16:12 **cambiar_velocidad.h** Page 1/1

```
1  #ifndef __CAMBIAR_VELOCIDAD_H__
2  #define __CAMBIAR_VELOCIDAD_H__
3
4  #include "transformacion.h"
5
6  #include "../movimiento/velocidad.h"
7
8  // Forward declaration
9  class Entidad;
10
11 class CambiarVelocidad : public Transformacion {
12     private:
13     Entidad& entidad_;
14     Velocidad velocidad_;
15
16     public:
17     CambiarVelocidad(Fisicas& unasFisicas, Entidad& unaEntidad, Velocidad& unaVe
   locidad);
18     virtual ~CambiarVelocidad();
19     virtual void aplicar() override;
20 };
21
22 #endif
```

jun 11, 19 16:12 **cambiar_velocidad.cpp** Page 1/1

```
1  #include "cambiar_velocidad.h"
2
3  #include "../fisicas.h"
4
5  CambiarVelocidad::CambiarVelocidad(Fisicas& unasFisicas, Entidad& unaEntidad, Ve
   locidad& unaVelocidad) :
6      Transformacion(unasFisicas),
7      entidad_(unaEntidad),
8      velocidad_(unaVelocidad) {
9  }
10
11 CambiarVelocidad::~CambiarVelocidad() {
12 }
13
14 void CambiarVelocidad::aplicar() {
15     fisicas_.ejecutarCambiarVelocidad(entidad_, velocidad_);
16 }
17
```

jun 11, 19 16:12 **agregar_entidad.h** Page 1/1

```
#ifndef __AGREGAR_ENTIDAD_H__
#define __AGREGAR_ENTIDAD_H__

#include "transformacion.h"

#include "../movimiento/posicion.h"
#include "../formas/forma.h"

// Forward Declaration
class Entidad;

class AgregarEntidad : public Transformacion {
    private:
    Entidad& entidad_;
    Posicion posicion_;
    Forma forma_;

    public:
    AgregarEntidad(Fisicas& unasFisicas, Entidad& unaEntidad, Posicion& unaPosic
ion, Forma& unaForma);
    virtual ~AgregarEntidad();
    virtual void aplicar() override;
};

#endif
```

jun 11, 19 16:12 **agregar_entidad.cpp** Page 1/1

```
#include "agregar_entidad.h"

#include "../fisicas.h"

AgregarEntidad::AgregarEntidad(Fisicas& unasFisicas, Entidad& unaEntidad, Posici
on& unaPosicion, Forma& unaForma) :
    Transformacion(unasFisicas),
    entidad_(unaEntidad),
    posicion_(unaPosicion),
    forma_(unaForma) {
}

AgregarEntidad::~AgregarEntidad() {
}

void AgregarEntidad::aplicar() {
    fisicas_.ejecutarAgregar(entidad_, posicion_, forma_);
}
```

jun 11, 19 16:12 **velocidad.h** Page 1/1

```
1  #ifndef __VELOCIDAD_H__
2  #define __VELOCIDAD_H__
3
4  class Velocidad {
5      private:
6      float coordenadaX_;
7      float coordenadaY_;
8
9      public:
10     Velocidad(float x, float y);
11     ~Velocidad();
12     float x();
13     float y();
14 };
15
16 #endif
```

jun 11, 19 16:12 **velocidad.cpp** Page 1/1

```
1  #include "velocidad.h"
2
3  Velocidad::Velocidad(float x, float y) :
4      coordenadaX_(x),
5      coordenadaY_(y) {
6  }
7
8  Velocidad::~Velocidad() {
9  }
10
11 float Velocidad::x() {
12     return coordenadaX_;
13 }
14
15 float Velocidad::y() {
16     return coordenadaY_;
17 }
```

jun 11, 19 16:12 **rotacion.h** Page 1/1

```
#ifndef __ROTACION_H__
#define __ROTACION_H__

#include <cstddef>

class Rotacion {
    private:
    size_t angulo_;

    public:
    Rotacion(size_t unAnguloEnGrados);
    ~Rotacion();
    float anguloRadianes();
    size_t anguloGrados();
};

#endif
```

jun 11, 19 16:12 **rotacion.cpp** Page 1/1

```
#include "rotacion.h"

#define DECIMAL_A_RADIANES 0.0174533f

Rotacion::Rotacion(size_t unAnguloEnGrados) :
    angulo_(unAnguloEnGrados) {
}

Rotacion::~Rotacion() {
}

float Rotacion::anguloRadianes() {
    return angulo_ * DECIMAL_A_RADIANES;
}

size_t Rotacion::anguloGrados() {
    return angulo_;
}
```

jun 11, 19 16:12 **posicion.h** Page 1/1

```
1  #ifndef __POSICION_H__
2  #define __POSICION_H__
3
4  class Posicion {
5      private:
6      float x_;
7      float y_;
8
9      public:
10     Posicion(float coordenadaX, float coordenadaY);
11     float x() const;
12     float y() const;
13     Posicion operator+(const Posicion& otra);
14 };
15
16 #endif
```

jun 11, 19 16:12 **posicion.cpp** Page 1/1

```
1  #include "posicion.h"
2
3  Posicion::Posicion(float coordenadaX, float coordenadaY) :
4      x_(coordenadaX),
5      y_(coordenadaY) {
6  }
7
8  float Posicion::x() const {
9      return x_;
10 }
11
12 float Posicion::y() const {
13     return y_;
14 }
15
16 Posicion Posicion::operator+(const Posicion& otra) {
17     return Posicion(x_ + otra.x(), y_ + otra.y());
18 }
```

jun 11, 19 16:12 **direccion.h** Page 1/1

```
1  #ifndef __DIRECCION_H__
2  #define __DIRECCION_H__
3
4  #include "Box2D/Box2D.h"
5
6  // Forward declaration
7  class Rotacion;
8
9  class Direccion {
10     private:
11     float x_;
12     float y_;
13
14     public:
15     Direccion(float x, float y);
16     Direccion copiar();
17     b2Vec2 transformar(b2Vec2& unaVelocidad);
18     void rotar(Rotacion& r);
19 };
20
21 #endif
```

jun 11, 19 16:12 **direccion.cpp** Page 1/1

```
1  #include "direccion.h"
2  #include "rotacion.h"
3
4  Direccion::Direccion(float x, float y) :
5      x_(x),
6      y_(y) {
7  }
8
9  Direccion Direccion::copiar() {
10     Direccion tmp(x_, y_);
11     return tmp;
12 }
13
14 b2Vec2 Direccion::transformar(b2Vec2& unaVelocidad) {
15     float norma = unaVelocidad.Length();
16     return b2Vec2(x_ * norma, y_ * norma);
17 }
18
19 void Direccion::rotar(Rotacion& r) {
20     int a = 0;
21     int b = -1;
22     int c = 1;
23     int d = 0;
24
25     for (size_t i = 0; i < r.anguloGrados() / 90; i++) {
26         x_ = (a*x_ + b*y_);
27         y_ = (c*x_ + d*y_);
28     }
29 }
```

jun 11, 19 16:12 **forma.h** Page 1/1

```
1  #ifndef __FORMA_H__
2  #define __FORMA_H__
3
4  class Forma {
5      private:
6      float ancho_;
7      float alto_;
8
9      public:
10     Forma(float unAncho, float unAlto);
11     float ancho();
12     float alto();
13 };
14
15 #endif
```

jun 11, 19 16:12 **forma.cpp** Page 1/1

```
1  #include "forma.h"
2
3  Forma::Forma(float unAncho, float unAlto) :
4      ancho_(unAncho),
5      alto_(unAlto) {
6  }
7
8  float Forma::ancho() {
9      return ancho_;
10 }
11
12 float Forma::alto() {
13     return alto_;
14 }
15
```

jun 11, 19 16:12 **fisicas.h** Page 1/1

```
#ifndef __FISICAS_H__
#define __FISICAS_H__

#include <map>
#include <queue>
#include <memory>

#include "Box2D/Box2D.h"

#include "contactos.h"

// Forward declaration
class Forma;
class Bloque;
class Superficie;

class Entidad;
//class BloqueMetalCuadrado;
//class BloquePiedra;
//class Jugador;
//class DisparoPortal;
class Posicion;
class Velocidad;
class Rotacion;
class Transformacion;

class Fisicas {
    private:
    b2Vec2 gravedad_;
    b2World* mundoBox2D_;
    ContactListener contactListener_;
    std::map<int, b2Body*> colisionables_;
    std::queue<std::shared_ptr<Transformacion>> transformaciones_;

    public:
    Fisicas();
    ~Fisicas();
    void agregarBloqueRectangular(Bloque& unBloque, Posicion& unaPosicion, Forma & forma);
    void agregarBloqueTriangular(Bloque& unBloque, Posicion& unaPosicion, Forma& forma, Rotacion& r);
    void agregarSuperficie(Superficie& superficie, Posicion& posicion, Forma& forma, Rotacion& r);

    void agregarEntidad(Entidad& entidad, Posicion& posicion, Forma& forma);
    void ejecutarAgregar(Entidad& entidad, Posicion& posicion, Forma& forma);

    void cambiarVelocidad(Entidad& entidad, Velocidad& velocidad);
    void ejecutarCambiarVelocidad(Entidad& entidad, Velocidad& velocidad);
    //void agregarBloqueMetalCuadrado(BloqueMetalCuadrado& bloque, Posicion& posicion);
    //void agregarBloquePiedra(BloquePiedra& bloque, Posicion& posicion);
    //void agregarJugador(Jugador& jugador, Posicion& posicion);
    //void agregarDisparoPortal(DisparoPortal& disparo, Posicion& posicion);

    void aplicarTransformaciones();
    void step();

    // Testing
    void setMundo(b2World* mundoCreado) {mundoBox2D_ = mundoCreado; mundoBox2D_->SetContactListener(&contactListener_);}
};

#endif
```

jun 11, 19 16:12 **fisicas.cpp** Page 1/3

```
#include "fisicas.h"

#include <vector>

#include "../../server_config.h"

#include "../superficies/superficie.h"
#include "../superficies/bloque.h"
//#include "../entidades/bloque_metal_cuadrado.h"

#include "../entidades/entidad.h"

#include "movimiento/posicion.h"
#include "movimiento/velocidad.h"
#include "movimiento/rotacion.h"
#include "formas/forma.h"

#include "transformaciones/transformacion.h"
#include "transformaciones/agregar_entidad.h"
#include "transformaciones/cambiar_velocidad.h"

//#define PI 3.14159265358979323846f

Fisicas::Fisicas() :
    gravedad_(CONFIG.GRAVEDAD_X, CONFIG.GRAVEDAD_Y),
    mundoBox2D_(new b2World(gravedad_)) {
    mundoBox2D_->SetContactListener(&contactListener_);
}

Fisicas::~Fisicas() {
    //delete mundoBox2D_;
}

void Fisicas::step() {
    mundoBox2D_->Step(CONFIG.TIME_STEP, CONFIG.VELOCITY_ITERATIONS, CONFIG.POSITION_ITERATIONS);
    aplicarTransformaciones();
}

void Fisicas::agregarBloqueRectangular(Bloque& unBloque, Posicion& unaPosicion, Forma& unaForma) {

    b2BodyDef b2CuerpoDef;
    b2CuerpoDef.type = b2_staticBody;
    b2CuerpoDef.position.Set(unaPosicion.x(), unaPosicion.y());
  b2CuerpoDef.userData = &unBloque;
    b2Body* b2Cuerpo = mundoBox2D_->CreateBody(&b2CuerpoDef);

    b2PolygonShape b2FormaCaja;
  b2FormaCaja.SetAsBox(unaForma.ancho(), unaForma.alto());
  b2FixtureDef b2Caracteristicas;
  b2Caracteristicas.shape = &b2FormaCaja;
  b2Cuerpo->CreateFixture(&b2Caracteristicas);

    colisionables_[unBloque.uuid()] = b2Cuerpo;
}

void Fisicas::agregarBloqueTriangular(Bloque& unBloque, Posicion& unaPosicion, Forma& unaForma, Rotacion& r) {

    b2BodyDef b2CuerpoDef;
    b2CuerpoDef.type = b2_staticBody;
    b2CuerpoDef.position.Set(unaPosicion.x(), unaPosicion.y());
  b2CuerpoDef.userData = &unBloque;
    b2Body* b2Cuerpo = mundoBox2D_->CreateBody(&b2CuerpoDef);

```

```
64      b2Vec2 vertices[3];
65      vertices[0].Set(0.0f, 0.0f);
66      vertices[1].Set(2 * unaForma.ancho(), 0.0f);
67      vertices[2].Set(0.0f, 2 * unaForma.alto());
68
69      std::vector<b2Vec2> deltas;
70      deltas.push_back(b2Vec2(-1.0f, -1.0f));
71      deltas.push_back(b2Vec2(1.0f, -1.0f));
72      deltas.push_back(b2Vec2(1.0f, 1.0f));
73      deltas.push_back(b2Vec2(-1.0f, 1.0f));
74
75      int indice_delta = 0;
76
77      for (size_t i = 0; i < r.anguloGrados() / 90; i++) {
78          indice_delta++;
79      }
80
81      int32 count = 3;
82      b2PolygonShape b2Triangulo;
83      b2Triangulo.Set(vertices, count);
84
85    b2FixtureDef b2Caracteristicas;
86    b2Caracteristicas.shape = &b2Triangulo;
87    b2Cuerpo→CreateFixture(&b2Caracteristicas);
88      b2Cuerpo→SetTransform(b2Cuerpo→GetWorldCenter() + deltas[indice_delta], r.
    anguloRadianes());
89
90      colisionables_[unBloque.uuid()] = b2Cuerpo;
91  }
92
93  void Fisicas::agregarSuperficie(Superficie& unaSuperficie, Posicion& unaPosicion
    , Forma& unaForma, Rotacion& r) {
94      b2BodyDef b2CuerpoDef;
95      b2CuerpoDef.type = b2_staticBody;
96      b2CuerpoDef.position.Set(unaPosicion.x(), unaPosicion.y());
97    b2CuerpoDef.userData = &unaSuperficie;
98      b2Body* b2Cuerpo = mundoBox2D_→CreateBody(&b2CuerpoDef);
99
100     b2PolygonShape b2FormaCaja;
101   b2FormaCaja.SetAsBox(unaForma.ancho(), unaForma.alto());
102   b2FixtureDef b2Caracteristicas;
103   b2Caracteristicas.shape = &b2FormaCaja;
104     b2Caracteristicas.isSensor = true;
105   b2Cuerpo→CreateFixture(&b2Caracteristicas);
106
107     b2Cuerpo→SetTransform(b2Cuerpo→GetPosition(), r.anguloRadianes());
108
109     colisionables_[unaSuperficie.uuid()] = b2Cuerpo;
110 }
111
112 void Fisicas::agregarEntidad(Entidad& unaEntidad, Posicion& unaPosicion, Forma&
    unaForma) {
113     transformaciones_.push(std::shared_ptr<Transformacion>(new AgregarEntidad(*t
    his, unaEntidad, unaPosicion, unaForma)));
114 }
115
116 void Fisicas::ejecutarAgregar(Entidad& unaEntidad, Posicion& unaPosicion, Forma&
     unaForma) {
117     b2BodyDef b2CuerpoDef;
118     // Para simular mejor el movimiento
119   b2CuerpoDef.bullet = true;
120   b2CuerpoDef.type = b2_dynamicBody;
121   b2CuerpoDef.position.Set(unaPosicion.x(), unaPosicion.y());
122   b2CuerpoDef.userData = &unaEntidad;
123   b2Body* b2Cuerpo = mundoBox2D_→CreateBody(&b2CuerpoDef);
124
```



```
125     b2PolygonShape b2FormaCaja;
126   b2FormaCaja.SetAsBox(unaForma.ancho(), unaForma.alto());
127   b2FixtureDef b2Caracteristicas;
128   b2Caracteristicas.shape = &b2FormaCaja;
129   b2Cuerpo→CreateFixture(&b2Caracteristicas);
130
131   colisionables_[unaEntidad.uuid()] = b2Cuerpo;
132 }
133
134 void Fisicas::cambiarVelocidad(Entidad& unaEntidad, Velocidad& unaVelocidad) {
135     transformaciones_.push(std::shared_ptr<Transformacion>(new CambiarVelocidad(
    *this, unaEntidad, unaVelocidad)));
136 }
137
138 void Fisicas::ejecutarCambiarVelocidad(Entidad& unaEntidad, Velocidad& unaVeloci
    dad) {
139     b2Body* b2Entidad = colisionables_[unaEntidad.uuid()];
140     b2Vec2 nuevaVelocidad = b2Vec2(unaVelocidad.x(), unaVelocidad.y());
141     b2Entidad→SetLinearVelocity(nuevaVelocidad);
142 }
143
144 void Fisicas::aplicarTransformaciones() {
145     while (¬transformaciones_.empty()) {
146     std::shared_ptr<Transformacion> t = transformaciones_.front();
147     t→aplicar();
148     transformaciones_.pop();
149     }
150 }
```

jun 11, 19 16:12 **contactos.h** Page 1/1

```
1  #ifndef __CONTACTOS_H__
2  #define __CONTACTOS_H__
3
4  #include "Box2D/Box2D.h"
5
6  class ContactListener : public b2ContactListener {
7      public:
8      ContactListener();
9      virtual ~ContactListener();
10     virtual void PreSolve(b2Contact* contact, const b2Manifold* oldManifold) ove
   rride;
11     virtual void BeginContact(b2Contact* contact) override;
12     virtual void EndContact(b2Contact* contact) override;
13     virtual void PostSolve(b2Contact* contact, const b2ContactImpulse* impulse)
   override;
14 };
15
16 #endif
```

jun 11, 19 16:12 **contactos.cpp** Page 1/1

```
1  #include "contactos.h"
2
3  #include "../colisionable.h"
4
5  ContactListener::ContactListener() {
6  }
7
8  ContactListener::~ContactListener() {
9  }
10
11 void ContactListener::BeginContact(b2Contact* contact) {
12     Colisionable* colisionableA = contact→GetFixtureA()→GetBody()→GetUserData(
   );
13     Colisionable* colisionableB = contact→GetFixtureB()→GetBody()→GetUserData(
   );
14
15     if(¬colisionableA ∨ ¬colisionableB) {
16         return;
17     }
18
19     colisionableA→colisionarContra(*colisionableB);
20     colisionableB→colisionarContra(*colisionableA);
21 }
22
23 void ContactListener::EndContact(b2Contact* contact) {
24 }
25
26 void ContactListener::PreSolve(b2Contact* contact, const b2Manifold* oldManifold
   ) {
27
28 }
29
30 void ContactListener::PostSolve(b2Contact* contact, const b2ContactImpulse* impu
   lse) {
31
32 }
```

jun 11, 19 16:12 **jugador.h** Page 1/1

```h
#ifndef __JUGADOR_H__
#define __JUGADOR_H__

#include "entidad.h"

// Forward declaration
class Colisionable;

class Jugador : public Entidad {
    public:
    Jugador(Fisicas& unasFisicas);
    virtual ~Jugador();
    virtual void colisionarContra(Colisionable& otro) override;
};

#endif
```

jun 11, 19 16:12 **jugador.cpp** Page 1/1

```cpp
#include "jugador.h"

#include "../colisionable.h"

Jugador::Jugador(Fisicas& unasFisicas) :
    Entidad(unasFisicas) {
}

void Jugador::colisionarContra(Colisionable& otro) {
    otro.colisionarContra(*this);
}

Jugador::~Jugador() {
}
```

jun 11, 19 16:12 **entidad.h** Page 1/1

```
1  #ifndef __ENTIDAD_H__
2  #define __ENTIDAD_H__
3
4  #include "../colisionable.h"
5
6  class Entidad : public Colisionable {
7      public:
8      Entidad(Fisicas& unasFisicas);
9      virtual ~Entidad();
10 };
11
12 #endif
```

jun 11, 19 16:12 **entidad.cpp** Page 1/1

```
1  #include "entidad.h"
2
3  Entidad::Entidad(Fisicas& unasFisicas) :
4      Colisionable(unasFisicas) {
5  }
6
7  Entidad::~Entidad() {
8  }
```

jun 11, 19 16:12 **colisionable.h** Page 1/1

```
#ifndef __COLISIONABLE_H__
#define __COLISIONABLE_H__

#include "identificable.h"

// Forward declaration
class Jugador;
class DisparoPortal;
//class Roca;
//class DisparoPin;
//class BolaEnergia;

//class BloqueEmisor;
//class BloqueReceptor;
//class BloqueMetalCuadrado;
//class BloqueMetalTriangular;
//class BloquePiedra;
//class BarreraEnergia;
//class SuperficieBoton;
//class SuperficieCompuerta;
//class Portal;

class Fisicas;

class Colisionable : public Identificable {
    public:
    virtual void colisionarContra(Colisionable& otro) = 0;

    virtual void colisionarContra(Jugador& j);
    virtual void colisionarContra(DisparoPortal& d);
    //virtual void colisionarContra(Roca& r);
    //virtual void colisionarContra(DisparoPin& d);
    //virtual void colisionarContra(BolaEnergia& b);

    //virtual void colisionarContra(BloqueEmisor& b);
    //virtual void colisionarContra(BloqueReceptor& b);
    //virtual void colisionarContra(BloqueMetalCuadrado& b);
    //virtual void colisionarContra(BloqueMetalTriangular& b);
    //virtual void colisionarContra(BloquePiedra& b);
    //virtual void colisionarContra(BarreraEnergia& b);
    //virtual void colisionarContra(SuperficieBoton& s);
    //virtual void colisionarContra(SuperficieCompuerta& s);
    //virtual void colisionarContra(Portal& p);

    Colisionable(Fisicas& unasFisicas);
    virtual ~Colisionable();

    protected:
    Fisicas& fisicas_;
};

#endif
```

jun 11, 19 16:12 **colisionable.cpp** Page 1/1

```
#include "colisionable.h"

Colisionable::Colisionable(Fisicas& unasFisicas) :
    fisicas_(unasFisicas) {
}

Colisionable::~Colisionable() {
}

void Colisionable::colisionarContra(Jugador& j) {

}

void Colisionable::colisionarContra(DisparoPortal& d) {

}
```

jun 11, 19 16:12 **value_protected.h** Page 1/1

```
1  #ifndef __VALUE_H__
2  #define __VALUE_H__
3
4  #include <mutex>
5
6  template <typename T>
7  class ValueProtected {
8  private:
9      T valor_;
10     std::mutex mtx;
11     ValueProtected(const ValueProtected& otro) = delete;
12     ValueProtected(ValueProtected∧ otro) = delete;
13     ValueProtected& operator=(const ValueProtected& otro) = delete;
14     ValueProtected& operator=(ValueProtected∧ otro) = delete;
15
16 public:
17     explicit ValueProtected(T unValor);
18     ~ValueProtected();
19     T operator()();
20     void set(T unValor);
21 };
22
23 #endif
```

jun 11, 19 16:12 **value_protected.cpp** Page 1/1

```
1  #include "value_protected.h"
2
3  template <typename T>
4  ValueProtected<T>::ValueProtected(T unValor) :
5      valor_(unValor) {
6  }
7
8  template <typename T>
9  ValueProtected<T>::~ValueProtected() {
10 }
11
12 template <typename T>
13 T ValueProtected<T>::operator()() {
14     std::lock_guard<std::mutex> lck(mtx);
15     return valor_;
16 }
17
18 template <typename T>
19 void ValueProtected<T>::set(T unValor) {
20     std::lock_guard<std::mutex> lck(mtx);
21     valor_ = unValor;
22 }
23
24 template class ValueProtected<bool>;
```

jun 11, 19 16:12 **Thread.h** Page 1/1

```
1  #ifndef _THREAD_
2  #define _THREAD_
3
4  #include <thread>
5
6  class Thread {
7  private:
8    std::thread thread;
9
10 public:
11   Thread();
12
13   Thread(Thread∧ origen);
14
15   Thread& operator=(Thread∧ origen);
16
17   Thread(const Thread&) = delete;
18
19   Thread& operator=(const Thread&) = delete;
20
21   //Inicia la ejecucion en un thread nuevo
22   void iniciar();
23
24   //Hace un join del thread
25   virtual void cerrar();
26
27   //Ejecucion del thread a ser implementada por quien herede esta clase
28   virtual void ejecutar() = 0;
29
30   virtual ~Thread();
31 };
32
33 #endif
```

jun 11, 19 16:12 **Thread.cpp** Page 1/1

```
1  #include "Thread.h"
2
3  Thread::Thread() {}
4  Thread::~Thread() {}
5
6  Thread::Thread(Thread∧ origen) {
7      this→thread = std::move(origen.thread);
8  }
9  Thread& Thread::operator=(Thread∧ origen) {
10     this→thread = std::move(origen.thread);
11     return *this;
12 }
13
14 void Thread::iniciar() {
15   this→thread = std::thread(&Thread::ejecutar, this);
16 }
17
18 void Thread::cerrar() {
19   this→thread.join();
20 }
```

jun 11, 19 16:12 **Socket.h** Page 1/1

```
#ifndef SOCKET
#define SOCKET

#include <stddef.h>
#include <sys/types.h>
#include <string.h>
#include <string>
#include <cstdint>

#define SKT_OK 0
#define SKT_INVALIDO -1
#define SKT_FALLO_CONEXION -2
#define SKT_ERROR -3

class Socket {
private:
    int fd;
    explicit Socket(int nuevoFd);

public:
    //Construye el socket
    Socket();
    Socket(Socket∧ origen);
  Socket& operator=(Socket∧ origen);

    // Bindea un socket al puerto especificado, y lo pone a la
    // espera de conectar con clientes.
    // Se escucha a un maximo de 'maxClientes' a la vez.
    int vincularYEscuchar(const char* puerto, size_t maxClientes);

    //Conecta un socket cliente con un servidor.
    int conectar(const char* hostname, const char* puerto);

    // En caso de haber algun cliente tratando de conectarse, el servidor trata
    // de aceptar la conexion. De concretarse, devuelve un nuevo Socket
    // resultado de esa conexion
    Socket aceptar();

    //Envia un mensaje de un cierto largo especificado.
    int enviar(const void* buffer, size_t largo);

    //Recibe un mensaje de un cierto largo especificado.
    int recibir(void* buffer, size_t largo);

    //Apaga el socket, de forma que no se puedan recibir ni enviar mensajes.
    void shutdown();

    //Sobrecarga de operadores para recibir o enviar enteros.
  void enviarInt(int num);
    int recibirInt();

  //Destruye el socket
    ~Socket();
};

#endif
```

jun 11, 19 16:12 **Socket.cpp** Page 1/3

```
#define _POSIX_C_SOURCE 200112L
#include <errno.h>
#include <stddef.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>

#include <string>
#include <iostream>
#include <stdexcept>
#include "Socket.h"

#define CUATRO_BYTES 4

Socket::Socket() {
  int familia = AF_INET;      /* IPv4 (or AF_INET6 for IPv6)     */
  int tipo = SOCK_STREAM;     /* TCP  (or SOCK_DGRAM for UDP)    */
  int protocolo = 0;          /* Any protocol */
  int s = socket(familia, tipo, protocolo);

  // Si falla al crear el socket
  if (s ≡ SKT_INVALIDO)
    throw std::runtime_error("Fallo la creacion del socket");

  this→fd = s;
}

Socket::~Socket() {
  if (this→fd ≠ SKT_INVALIDO)
    close(this→fd);
}

Socket::Socket(int nuevoFd): fd(nuevoFd) {}

Socket::Socket(Socket∧ origen): fd(origen.fd) {
  origen.fd = SKT_INVALIDO;
}

Socket& Socket::operator=(Socket∧ origen) {
  this→fd = origen.fd;
  origen.fd = SKT_INVALIDO;
  return *this;
}

int Socket::vincularYEscuchar(const char* puerto, size_t maxClientes) {
  struct addrinfo hints;
  struct addrinfo *results, *ptr;
  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;          /* IPv4 (or AF_INET6 for IPv6)     */
  hints.ai_socktype = SOCK_STREAM;    /* TCP  (or SOCK_DGRAM for UDP)    */
  hints.ai_flags = AI_PASSIVE;        /* AI_PASSIVE for server           */

  if (getaddrinfo(NULL, puerto, &hints, &results) ≠ 0)
    return SKT_FALLO_CONEXION;

  ptr = results;
  int bindeado = 0;
  while (ptr ≠ NULL){
    if (bind(this→fd, ptr→ai_addr, ptr→ai_addrlen) ≥ 0) {
      bindeado = 1;
      break;
    }
    ptr = ptr→ai_next;
  }
  freeaddrinfo(results);
```

```
67
68    if (bindeado ≡ 0)
69      return SKT_FALLO_CONEXION;
70
71    if (listen(this→fd, maxClientes) ≡ -1){
72      ::shutdown(this→fd, SHUT_RDWR);
73      return SKT_ERROR;
74    }
75
76    return SKT_OK;
77  }
78
79
80  int Socket::conectar(const char* hostname, const char* puerto){
81    struct addrinfo hints;
82    struct addrinfo *results, *ptr;
83    memset(&hints, 0, sizeof(struct addrinfo));
84    hints.ai_family = AF_INET;          /* IPv4 (or AF_INET6 for IPv6)     */
85    hints.ai_socktype = SOCK_STREAM;    /* TCP  (or SOCK_DGRAM for UDP)    */
86    hints.ai_flags = 0;                 /* None                            */
87
88    if (getaddrinfo(hostname, puerto, &hints, &results) ≠ 0)
89      return SKT_FALLO_CONEXION;
90
91    ptr = results;
92    int conexion_establecida = 0;
93    while (ptr ≠ NULL){
94      if (connect(this→fd, ptr→ai_addr, ptr→ai_addrlen) ≥ 0){
95        conexion_establecida = 1;
96        break;
97      }
98      ptr = ptr→ai_next;
99    }
100   freeaddrinfo(results);
101
102   if (conexion_establecida ≡ 0)
103     return SKT_FALLO_CONEXION;
104
105   return SKT_OK;
106 }
107
108
109 Socket Socket::aceptar(){
110   int fd = accept(this→fd, NULL, NULL);
111   if (fd ≡ SKT_INVALIDO)
112     throw std::runtime_error("Fallo al aceptar una conexion\n");
113
114   Socket peerSocket(fd);
115   return std::move(peerSocket);
116 }
117
118
119 int Socket::enviar(const void* buffer, size_t largo){
120   int e = 0;
121   size_t enviado = 0;
122   const char* buf = (const char*) buffer;
123
124   while (enviado < largo){
125     e = send(this→fd, &buf[enviado],
126              largo − enviado, MSG_NOSIGNAL);
127     if (e ≡ 0){
128       return SKT_FALLO_CONEXION;
129     }
130     if (e < 0){
131       return SKT_ERROR;
132     }
```



```
133     enviado += e;
134   }
135   return enviado;
136 }
137
138
139 int Socket::recibir(void* buffer, size_t largo){
140   int r = 0;
141   size_t recibido = 0;
142   char* buf = (char*) buffer;
143   while (recibido < largo){
144     r = recv(this→fd, &buf[recibido],
145              largo − recibido, MSG_NOSIGNAL);
146     if (r ≡ 0){
147       return SKT_FALLO_CONEXION;
148     }
149     if (r < 0){
150       return SKT_ERROR;
151     }
152     recibido += r;
153   }
154   return recibido;
155 }
156
157
158 void Socket::shutdown(){
159   ::shutdown(this→fd, SHUT_RDWR);
160 }
161
162 void Socket::enviarInt(int num) {
163   this→enviar((char*) &num, CUATRO_BYTES);
164 }
165
166 int Socket::recibirInt() {
167   int num;
168   this→recibir((char*) &num, CUATRO_BYTES);
169   return num;
170 }
```

jun 11, 19 16:12 **Serializador.h** Page 1/1

```
#ifndef SERIALIZADOR_H
#define SERIALIZADOR_H

#include "Socket.h"
#include "Evento.h"

class Serializador {
public:
  Evento* recibirEvento(Socket& socket);
};

#endif // SERIALIZADOR_H

```

jun 11, 19 16:12 **Serializador.cpp** Page 1/1

```
#include "Serializador.h"
#include "Constantes.h"

Evento* Serializador::recibirEvento(Socket& socket) {
  int tipoEvento = socket.recibirInt();
  switch(tipoEvento) {
    case(EVENTO_PORTAL_AZUL): return new EventoPortalAzul(socket);
    case(EVENTO_PORTAL_NARANJA): return new EventoPortalNaranja(socket);
    case(EVENTO_PIN_TOOL): return new EventoPinTool(socket);
    case(EVENTO_MOVER): return new EventoMover(socket);
    case(EVENTO_CAMBIO_ESTADO): return new EventoCambioEstado(socket);
    case(EVENTO_FLIP): return new EventoFlip(socket);
    case(EVENTO_CORRER): return new EventoCorrer(socket);
    case(EVENTO_ELIMINAR_ITEM): return new EventoEliminarItem(socket);
    case(EVENTO_ROTACION): return new EventoRotacion(socket);
    case(EVENTO_RESET_PORTALES): return new EventoResetPortales(socket);
    case(EVENTO_SALTO): return new EventoSalto(socket);
    case(EVENTO_DEJAR_DE_MOVERSE): return new EventoDejarDeMoverse(socket);
    case(EVENTO_CREAR_ITEM): return new EventoCrearItem(socket);
    case(EVENTO_CREACION_PERSONAJE): return new EventoCreacionPersonaje(socket);
    case(EVENTO_INICIAR_PARTIDA): return new EventoIniciarPartida(socket);
    case(EVENTO_SUICIDIO): return new EventoSuicidio(socket);
    default: throw std::runtime_error("Error: se intento recuperar evento con tipo no definido.");
  }
}
```

jun 11, 19 16:12 **Lock.h** Page 1/1

```
1  #ifndef LOCK
2  #define LOCK
3
4  #include <mutex>
5
6  class Lock {
7  private:
8    std::mutex &m;
9
10 public:
11   //Crea el Lock, tomando el mutex
12   explicit Lock(std::mutex &m);
13
14   //Libera el mutex
15   ~Lock();
16
17   Lock(const Lock&) = delete;
18   Lock& operator=(const Lock&) = delete;
19   Lock(Lock∧) = delete;
20   Lock& operator=(Lock∧) = delete;
21 };
22
23 #endif
```

jun 11, 19 16:12 **Lock.cpp** Page 1/1

```
1  #include <mutex>
2
3  #include "Lock.h"
4
5  Lock::Lock(std::mutex &m) : m(m) {
6    m.lock();
7  }
8
9  Lock::~Lock() {
10   m.unlock();
11 }
```

jun 11, 19 16:12 **handler.h** Page 1/1

```
#ifndef __HANDLER_H__
#define __HANDLER_H__

// Forward declaration
class Evento;
class EventoCrearItem;
class EventoMover;
class EventoFlip;
class EventoCambioEstado;
class EventoEliminarItem;
class EventoRotacion;
class EventoCreacionPersonaje;

class Handler {
    public:
    virtual void manejar(Evento& evento) = 0;
    virtual void manejar(EventoCrearItem& evento) {}
  virtual void manejar(EventoMover& evento) {}
  virtual void manejar(EventoFlip& evento) {}
  virtual void manejar(EventoCambioEstado& evento) {}
  virtual void manejar(EventoEliminarItem& evento) {}
  virtual void manejar(EventoRotacion& evento) {}
  virtual void manejar(EventoCreacionPersonaje& evento) {}
};

#endif
```

jun 11, 19 16:12 **Evento.h** Page 1/3

```
#ifndef EVENTO
#define EVENTO

#include <map>

class Handler;
class Socket;

class Evento {
protected:
  int tipo;
public:
  std::map<std::string, int> atributos;
  virtual void enviarPorSocket(Socket& s) = 0;
  virtual void actualizar(Handler& handler) = 0;
  virtual ~Evento() {}
};

class EventoIniciarPartida : public Evento {
public:
  EventoIniciarPartida();
  EventoIniciarPartida(Socket& s);
  virtual void enviarPorSocket(Socket& s);
  virtual void actualizar(Handler& handler) override {}
  virtual ~EventoIniciarPartida() {}
};

class EventoCreacionPersonaje : public Evento {
public:
  EventoCreacionPersonaje(int idPersonaje);
  EventoCreacionPersonaje(Socket& s);
  virtual void enviarPorSocket(Socket& s);
  virtual void actualizar(Handler& handler) override;
  virtual ~EventoCreacionPersonaje() {}
};

//TODO: cambiar a direccion
class EventoPortalAzul : public Evento {
public:
  EventoPortalAzul(int x, int y);
  EventoPortalAzul(Socket& s);
  virtual void enviarPorSocket(Socket& s);
  virtual void actualizar(Handler& handler) override {}
  virtual ~EventoPortalAzul() {}
};
//TODO: cambiar a direccion
class EventoPortalNaranja : public Evento {
public:
  EventoPortalNaranja(int x, int y);
  EventoPortalNaranja(Socket& s);
  virtual void enviarPorSocket(Socket& s);
  virtual void actualizar(Handler& handler) override {}
  virtual ~EventoPortalNaranja() {}
};

class EventoCrearItem : public Evento {
public:
  EventoCrearItem(int idItem, int x, int y, int angulo);
  EventoCrearItem(Socket& s);
  virtual void enviarPorSocket(Socket& s);
  virtual void actualizar(Handler& handler) override;
  virtual ~EventoCrearItem() {}
};

class EventoResetPortales : public Evento {
public:
```

```
67    EventoResetPortales(int idLanzador);
68    EventoResetPortales(Socket& s);
69    virtual void enviarPorSocket(Socket& s);
70    virtual void actualizar(Handler& handler) override {}
71    virtual ~EventoResetPortales() {}
72  };
73
74  class EventoDejarDeMoverse : public Evento {
75  public:
76    EventoDejarDeMoverse(int idLanzador);
77    EventoDejarDeMoverse(Socket& s);
78    virtual void enviarPorSocket(Socket& s);
79    virtual void actualizar(Handler& handler) override {}
80    virtual ~EventoDejarDeMoverse() {}
81  };
82
83  class EventoMover : public Evento {
84  public:
85    EventoMover(int x, int y, int idLanzador);
86    EventoMover(Socket& s);
87    virtual void enviarPorSocket(Socket& s);
88    virtual void actualizar(Handler& handler) override;
89    virtual ~EventoMover() {}
90  };
91
92  class EventoCorrer : public Evento {
93  public:
94    EventoCorrer(int direccion, int idLanzador);
95    EventoCorrer(Socket& s);
96    virtual void enviarPorSocket(Socket& s);
97    virtual void actualizar(Handler& handler) override {}
98    virtual ~EventoCorrer() {}
99  };
100
101 class EventoPinTool : public Evento {
102 public:
103   EventoPinTool(int x, int y);
104   EventoPinTool(Socket& s);
105   virtual void enviarPorSocket(Socket& s);
106   virtual void actualizar(Handler& handler) override {}
107   virtual ~EventoPinTool() {}
108 };
109
110 class EventoSuicidio : public Evento {
111 public:
112   EventoSuicidio(int idLanzador);
113   EventoSuicidio(Socket& s);
114   virtual void enviarPorSocket(Socket& s);
115   virtual void actualizar(Handler& handler) override {}
116   virtual ~EventoSuicidio() {}
117 };
118
119 class EventoSalto : public Evento {
120 public:
121   EventoSalto(int idLanzador);
122   EventoSalto(Socket& s);
123   virtual void enviarPorSocket(Socket& s);
124   virtual void actualizar(Handler& handler) override {}
125   virtual ~EventoSalto() {}
126 };
127
128 class EventoFlip : public Evento {
129 public:
130   EventoFlip(int flip, int idItem);
131   EventoFlip(Socket& s);
132   virtual void enviarPorSocket(Socket& s);
```



```
133   virtual void actualizar(Handler& handler) override;
134   virtual ~EventoFlip() {}
135 };
136
137 class EventoCambioEstado : public Evento {
138 public:
139   EventoCambioEstado(int estado, int idItem);
140   EventoCambioEstado(Socket& s);
141   virtual void enviarPorSocket(Socket& s);
142   virtual void actualizar(Handler& handler) override;
143   virtual ~EventoCambioEstado() {}
144 };
145
146 class EventoEliminarItem : public Evento {
147 public:
148   EventoEliminarItem(int idItem);
149   EventoEliminarItem(Socket& s);
150   virtual void enviarPorSocket(Socket& s);
151   virtual void actualizar(Handler& handler) override;
152   virtual ~EventoEliminarItem() {}
153 };
154
155 class EventoRotacion : public Evento {
156 public:
157   EventoRotacion(int angulo, int idItem);
158   EventoRotacion(Socket& s);
159   virtual void enviarPorSocket(Socket& s);
160   virtual void actualizar(Handler& handler) override;
161   virtual ~EventoRotacion() {}
162 };
163
164 #endif
```

```cpp
#include "Evento.h"
#include "Constantes.h"
#include "Socket.h"
#include "handler.h"

#include <iostream>


EventoIniciarPartida::EventoIniciarPartida() {
  tipo = EVENTO_INICIAR_PARTIDA;
}
EventoIniciarPartida::EventoIniciarPartida(Socket& s) {
  tipo = EVENTO_INICIAR_PARTIDA;
}
void EventoIniciarPartida::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
}


EventoCreacionPersonaje::EventoCreacionPersonaje(int idPersonaje) {
  tipo = EVENTO_CREACION_PERSONAJE;
  atributos["idPersonaje"] = idPersonaje;
}
EventoCreacionPersonaje::EventoCreacionPersonaje(Socket& s) {
  tipo = EVENTO_CREACION_PERSONAJE;
  atributos["idPersonaje"] = s.recibirInt();
}
void EventoCreacionPersonaje::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["idPersonaje"]);
}


EventoPortalAzul::EventoPortalAzul(int x, int y) {
  tipo = EVENTO_PORTAL_AZUL;
  atributos["x"] = x;
  atributos["y"] = y;
}
EventoPortalAzul::EventoPortalAzul(Socket& s) {
  tipo = EVENTO_PORTAL_AZUL;
  atributos["x"] = s.recibirInt();
  atributos["y"] = s.recibirInt();
}
void EventoPortalAzul::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["x"]);
  s.enviarInt(atributos["y"]);
}


EventoPortalNaranja::EventoPortalNaranja(int x, int y) {
  tipo = EVENTO_PORTAL_NARANJA;
  atributos["x"] = x;
  atributos["y"] = y;
}
EventoPortalNaranja::EventoPortalNaranja(Socket& s) {
  tipo = EVENTO_PORTAL_NARANJA;
  atributos["x"] = s.recibirInt();
  atributos["y"] = s.recibirInt();
}
void EventoPortalNaranja::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["x"]);
  s.enviarInt(atributos["y"]);
}

```



```cpp

EventoCrearItem::EventoCrearItem(int idItem, int x, int y, int angulo) {
  tipo = EVENTO_CREAR_ITEM;
  atributos["idItem"] = idItem;
  atributos["x"] = x;
  atributos["y"] = y;
  atributos["angulo"] = angulo;
}
EventoCrearItem::EventoCrearItem(Socket& s) {
  tipo = EVENTO_CREAR_ITEM;
  atributos["idItem"] = s.recibirInt();
  atributos["x"] = s.recibirInt();
  atributos["y"] = s.recibirInt();
  atributos["angulo"] = s.recibirInt();
}
void EventoCrearItem::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["idItem"]);
  s.enviarInt(atributos["x"]);
  s.enviarInt(atributos["y"]);
  s.enviarInt(atributos["angulo"]);
}


EventoResetPortales::EventoResetPortales(int idLanzador) {
  tipo = EVENTO_RESET_PORTALES;
  atributos["idLanzador"] = idLanzador;
}
EventoResetPortales::EventoResetPortales(Socket& s) {
  tipo = EVENTO_RESET_PORTALES;
  atributos["idLanzador"] = s.recibirInt();
}
void EventoResetPortales::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["idLanzador"]);
}


EventoDejarDeMoverse::EventoDejarDeMoverse(int idLanzador) {
  tipo = EVENTO_DEJAR_DE_MOVERSE;
  atributos["idLanzador"] = idLanzador;
}
EventoDejarDeMoverse::EventoDejarDeMoverse(Socket& s) {
  tipo = EVENTO_DEJAR_DE_MOVERSE;
  atributos["idLanzador"] = s.recibirInt();
}
void EventoDejarDeMoverse::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["idLanzador"]);
}


EventoCorrer::EventoCorrer(int direccion, int idLanzador) {
  tipo = EVENTO_CORRER;
  atributos["direccion"] = direccion;
  atributos["idLanzador"] = idLanzador;
}
EventoCorrer::EventoCorrer(Socket& s) {
  tipo = EVENTO_CORRER;
  atributos["direccion"] = s.recibirInt();
  atributos["idLanzador"] = s.recibirInt();
}
void EventoCorrer::enviarPorSocket(Socket& s) {
  s.enviarInt(tipo);
  s.enviarInt(atributos["direccion"]);
  s.enviarInt(atributos["idLanzador"]);
```

```
133 }
134
135
136 EventoMover::EventoMover(int x, int y, int idLanzador) {
137   tipo = EVENTO_MOVER;
138   atributos["x"] = x;
139   atributos["y"] = y;
140   atributos["idLanzador"] = idLanzador;
141 }
142 EventoMover::EventoMover(Socket& s) {
143   tipo = EVENTO_MOVER;
144   atributos["x"] = s.recibirInt();
145   atributos["y"] = s.recibirInt();
146   atributos["idLanzador"] = s.recibirInt();
147 }
148 void EventoMover::enviarPorSocket(Socket& s) {
149   s.enviarInt(tipo);
150   s.enviarInt(atributos["x"]);
151   s.enviarInt(atributos["y"]);
152   s.enviarInt(atributos["idLanzador"]);
153 }
154
155
156 EventoPinTool::EventoPinTool(int x, int y) {
157   tipo = EVENTO_PIN_TOOL;
158   atributos["x"] = x;
159   atributos["y"] = y;
160 }
161 EventoPinTool::EventoPinTool(Socket& s) {
162   tipo = EVENTO_PIN_TOOL;
163   atributos["x"] = s.recibirInt();
164   atributos["y"] = s.recibirInt();
165 }
166 void EventoPinTool::enviarPorSocket(Socket& s) {
167   s.enviarInt(tipo);
168   s.enviarInt(atributos["x"]);
169   s.enviarInt(atributos["y"]);
170 }
171
172
173 EventoSalto::EventoSalto(int idLanzador) {
174   tipo = EVENTO_SALTO;
175   atributos["idLanzador"] = idLanzador;
176 }
177 EventoSalto::EventoSalto(Socket& s) {
178   tipo = EVENTO_SALTO;
179   atributos["idLanzador"] = s.recibirInt();
180 }
181 void EventoSalto::enviarPorSocket(Socket& s) {
182   s.enviarInt(tipo);
183   s.enviarInt(atributos["idLanzador"]);
184 }
185
186
187 EventoSuicidio::EventoSuicidio(int idLanzador) {
188   tipo = EVENTO_SUICIDIO;
189   atributos["idLanzador"] = idLanzador;
190 }
191 EventoSuicidio::EventoSuicidio(Socket& s) {
192   tipo = EVENTO_SUICIDIO;
193   atributos["idLanzador"] = s.recibirInt();
194 }
195 void EventoSuicidio::enviarPorSocket(Socket& s) {
196   s.enviarInt(tipo);
197   s.enviarInt(atributos["idLanzador"]);
198 }
```



```
199
200
201 EventoFlip::EventoFlip(int flip, int idItem) {
202   tipo = EVENTO_FLIP;
203   atributos["flip"] = flip;
204   atributos["idItem"] = idItem;
205 }
206 EventoFlip::EventoFlip(Socket& s) {
207   tipo = EVENTO_FLIP;
208   atributos["flip"] = s.recibirInt();
209   atributos["idItem"] = s.recibirInt();
210 }
211 void EventoFlip::enviarPorSocket(Socket& s) {
212   s.enviarInt(tipo);
213   s.enviarInt(atributos["flip"]);
214   s.enviarInt(atributos["idItem"]);
215 }
216
217
218 EventoCambioEstado::EventoCambioEstado(int estado, int idItem) {
219   tipo = EVENTO_CAMBIO_ESTADO;
220   atributos["estado"] = estado;
221   atributos["idItem"] = idItem;
222 }
223 EventoCambioEstado::EventoCambioEstado(Socket& s) {
224   tipo = EVENTO_CAMBIO_ESTADO;
225   atributos["estado"] = s.recibirInt();
226   atributos["idItem"] = s.recibirInt();
227 }
228 void EventoCambioEstado::enviarPorSocket(Socket& s) {
229   s.enviarInt(tipo);
230   s.enviarInt(atributos["estado"]);
231   s.enviarInt(atributos["idItem"]);
232 }
233
234
235 EventoEliminarItem::EventoEliminarItem(int idItem) {
236   tipo = EVENTO_ELIMINAR_ITEM;
237   atributos["idItem"] = idItem;
238 }
239 EventoEliminarItem::EventoEliminarItem(Socket& s) {
240   tipo = EVENTO_ELIMINAR_ITEM;
241   atributos["idItem"] = s.recibirInt();
242 }
243 void EventoEliminarItem::enviarPorSocket(Socket& s) {
244   s.enviarInt(tipo);
245   s.enviarInt(atributos["idItem"]);
246 }
247
248
249 EventoRotacion::EventoRotacion(int angulo, int idItem) {
250   tipo = EVENTO_ROTACION;
251   atributos["angulo"] = angulo;
252   atributos["idItem"] = idItem;
253 }
254 EventoRotacion::EventoRotacion(Socket& s) {
255   tipo = EVENTO_ROTACION;
256   atributos["angulo"] = s.recibirInt();
257   atributos["idItem"] = s.recibirInt();
258 }
259 void EventoRotacion::enviarPorSocket(Socket& s) {
260   s.enviarInt(tipo);
261   s.enviarInt(atributos["angulo"]);
262   s.enviarInt(atributos["idItem"]);
263 }
264
```

jun 11, 19 16:12 **Evento.cpp** Page 5/5

```
265 void EventoMover::actualizar(Handler& handler) {handler.manejar(*this);}
266 void EventoFlip::actualizar(Handler& handler) {handler.manejar(*this);}
267 void EventoCambioEstado::actualizar(Handler& handler) {handler.manejar(*this);}
268 void EventoEliminarItem::actualizar(Handler& handler) {handler.manejar(*this);}
269 void EventoRotacion::actualizar(Handler& handler) {handler.manejar(*this);}
270 void EventoCrearItem::actualizar(Handler& handler) {handler.manejar(*this);}
271 void EventoCreacionPersonaje::actualizar(Handler& handler) {handler.manejar(*thi
    s);}
```

jun 11, 19 16:12 **Conversor.h** Page 1/1

```
1  #ifndef CONVERSOR_H
2  #define CONVERSOR_H
3
4  class Conversor {
5  private:
6    int pixelesPorBloque;
7  public:
8    Conversor(int pixelesPorBloque);
9    int bloqueAPixel(float coord);
10   float pixelABloque(int coord);
11 };
12
13 #endif // CONVERSOR_H
```

jun 11, 19 16:12 **Conversor.cpp** Page 1/1

```cpp
#include "Conversor.h"
#include <cmath>
#include <iostream>

Conversor::Conversor(int pixelesPorBloque) :
  pixelesPorBloque(pixelesPorBloque) {}

int Conversor::bloqueAPixel(float coord) {
  return round((pixelesPorBloque*coord) + pixelesPorBloque/2);
}

float Conversor::pixelABloque(int coord) {
  return ((float) (coord - pixelesPorBloque/2) / pixelesPorBloque);
}
```

jun 11, 19 16:12 **Constantes.h** Page 1/1

```cpp
#define ESTADO_IDLE 0
#define ESTADO_CORRIENDO 1
#define ESTADO_DISPARANDO 2
#define ESTADO_MUERTO 3
#define ESTADO_SALTANDO 4

#define ESCENA_MENU 50
#define ESCENA_SALA 51
#define ESCENA_JUEGO 52

#define ID_BLOQUE_PIEDRA 100
#define ID_BLOQUE_METAL 101
#define ID_BLOQUE_DIAGONAL_METAL 102
#define ID_PIEDRA_MOVIL 104
#define ID_ACIDO 105
#define ID_BOTON 106
#define ID_EMISOR 107
#define ID_RECEPTOR 108
#define ID_PUERTA 109
#define ID_BARRERA_ENERGIA 110
#define ID_PERSONAJE_1 111
#define ID_PERSONAJE_2 112
#define ID_PERSONAJE_3 113
#define ID_PERSONAJE_4 114
#define ID_TORTA 115
#define ID_BOLA_ENERGIA 116
#define ID_PORTAL_AZUL 117
#define ID_PORTAL_NARANJA 118
#define ID_PIN_TOOL 119
#define ID_IMAGEN_MENU 120

#define EVENTO_PORTAL_AZUL 200
#define EVENTO_PORTAL_NARANJA 201
#define EVENTO_SALIR 202
#define EVENTO_PIN_TOOL 203
#define EVENTO_MOVER 204
#define EVENTO_FLIP 205
#define EVENTO_CAMBIO_ESTADO 206
#define EVENTO_CORRER 207
#define EVENTO_ELIMINAR_ITEM 208
#define EVENTO_ROTACION 209
#define EVENTO_SALTO 210
#define EVENTO_RESET_PORTALES 211
#define EVENTO_DEJAR_DE_MOVERSE 212
#define EVENTO_CREAR_ITEM 213
#define EVENTO_CREACION_PERSONAJE 214
#define EVENTO_INICIAR_PARTIDA 215
#define EVENTO_SUICIDIO 216

#define EFECTO_DISPARO 300
#define EFECTO_SALTO 301

#define DERECHA 1000
#define IZQUIERDA 1001
#define PRESIONADO 1002
#define NO_PRESIONADO 1003
#define ABIERTA 1004
#define CERRADA 1005
```

jun 11, 19 16:12 **Cola.h** Page 1/1

```
1  #ifndef COLA
2  #define COLA
3
4  #include <queue>
5
6  template <class T>
7  class Cola {
8  private:
9    std::queue<T> elementos;
10 public:
11   Cola() = default;
12   void put(T elem);
13   bool get(T& elem);
14 };
15
16 #endif
```

jun 11, 19 16:12 **Cola.cpp** Page 1/1

```
1  #include "Cola.h"
2  #include "Evento.h"
3
4  template<class T>
5  void Cola<T>::put(T elem) {
6    elementos.push(elem);
7  }
8
9  template<class T>
10 bool Cola<T>::get(T& elem) {
11   if (elementos.empty()) {
12     return false;
13   } else {
14     elem = elementos.front();
15     elementos.pop();
16     return true;
17   }
18 }
19
20 template class Cola<Evento*>;
```

jun 11, 19 16:12 **cola_bloqueante.h** Page 1/1

```
#include "value_protected.h"

#ifndef __COLA_CLIENTES_H__
#define __COLA_CLIENTES_H__

#include <mutex>
#include <condition_variable>
#include <queue>

// Cola protegida. Permite agregar y obtener elementos, mientras
// la cola estÃ© activa. Para detener y desactivar la cola, debe llamarse
// al mÃ©todo detener().
template <class T>
class ColaBloqueante {
    private:
    std::mutex mtx_;
    std::queue<T> elementos_;
    std::condition_variable cond_;
    ValueProtected<bool> detenida_;

    ColaBloqueante(ColaBloqueante∧ otra) = delete;

    ColaBloqueante(const ColaBloqueante& otra) = delete;

    ColaBloqueante& operator=(const ColaBloqueante& otra) = delete;

    ColaBloqueante& operator=(ColaBloqueante∧ otra) = delete;

    public:
    ColaBloqueante();

    ~ColaBloqueante();

    void put(T& unElemento);

    bool get(T& unElemento);

    void detener();
};

#endif
```

jun 11, 19 16:12 **cola_bloqueante.cpp** Page 1/1

```
#include "cola_bloqueante.h"
#include "Evento.h"

template <class T>
ColaBloqueante<T>::ColaBloqueante() :
    detenida_(false) {
}
template <class T>
ColaBloqueante<T>::~ColaBloqueante() {
}

template <class T>
void ColaBloqueante<T>::put(T& unElemento) {
    std::lock_guard<std::mutex> lck(mtx_);
    elementos_.push(std::move(unElemento));
    cond_.notify_one();
}

template <class T>
bool ColaBloqueante<T>::get(T& unElemento) {
    std::unique_lock<std::mutex> lck(mtx_);
    cond_.wait(lck, [this]{return ¬elementos_.empty() ∨ detenida_() ;});
    if (detenida_()) {
        return false;
    }
    unElemento = std::move(elementos_.front());
    elementos_.pop();
    return true;
}

template <class T>
void ColaBloqueante<T>::detener() {
    detenida_.set(true);
    cond_.notify_all();
}

template class ColaBloqueante<Evento*>;

```

jun 11, 19 16:12 **VistaTorta.h** Page 1/1

```
1  #ifndef VISTA_TORTA
2  #define VISTA_TORTA
3
4  #include "VistaObjeto.h"
5
6  class VistaTorta : public VistaObjeto {
7  public:
8    VistaTorta(SdlTexture& tex);
9    virtual void dibujarEn(int x, int y);
10   virtual ~VistaTorta() {}
11 };
12
13 #endif
```

jun 11, 19 16:12 **VistaTorta.cpp** Page 1/1

```
1  #include "VistaTorta.h"
2
3  VistaTorta::VistaTorta(SdlTexture& tex) {
4    this→textura = tex;
5    Area srcArea(200, 85, 160, 135);
6    clips.push_back(srcArea);
7    tamanioHorizontal = 100;
8    tamanioVertical = 75;
9  }
10
11 void VistaTorta::dibujarEn(int x, int y) {
12   Area srcArea = clips.front();
13   Area destArea(xInicial + x − tamanioHorizontal/2,
14           yInicial + y − tamanioVertical/2,
15           tamanioHorizontal, tamanioVertical);;
16   textura.render(srcArea, destArea);
17 }
18
```

jun 11, 19 16:12 **VistaReceptor.h** Page 1/1

```
1  #ifndef VISTA_RECEPTOR
2  #define VISTA_RECEPTOR
3
4  #include "VistaObjeto.h"
5
6  class VistaReceptor : public VistaObjeto {
7  private:
8    int angulo;
9  public:
10   VistaReceptor(SdlTexture& tex, int angulo);
11   virtual void dibujarEn(int x, int y);
12   virtual ~VistaReceptor() {}
13 };
14
15 #endif
```

jun 11, 19 16:12 **VistaReceptor.cpp** Page 1/1

```
1  #include "VistaReceptor.h"
2
3  VistaReceptor::VistaReceptor(SdlTexture& tex, int angulo) {
4    this→textura = tex;
5    this→angulo = angulo;
6    Area srcArea(199, 0, 191, 191);
7    clips.push_back(srcArea);
8    tamanioHorizontal = 100;
9    tamanioVertical = 100;
10 }
11
12 void VistaReceptor::dibujarEn(int x, int y) {
13   Area srcArea = clips.front();
14   Area destArea(xInicial + x − tamanioHorizontal/2,
15           yInicial + y − tamanioVertical/2,
16           tamanioHorizontal, tamanioVertical);
17   textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
18 }
```

jun 11, 19 16:12 **VistaPuerta.h** Page 1/1

```
#ifndef VISTA_PUERTA
#define VISTA_PUERTA

#include "VistaObjeto.h"

class VistaPuerta : public VistaObjeto {
private:
  int estado;
public:
  VistaPuerta(SdlTexture& tex);
  virtual void asignarEstado(int estado) override;
  virtual void dibujarEn(int x, int y);
  virtual ~VistaPuerta() {}
};

#endif
```

jun 11, 19 16:12 **VistaPuerta.cpp** Page 1/1

```
#include "VistaPuerta.h"
#include "../Common/Constantes.h"

#define CANT_CLIPS 38

VistaPuerta::VistaPuerta(SdlTexture& tex) {
  frame = 0;
  this→textura = tex;
  this→estado = 0;

  //Abierta
  Area areaAbierta(1, 2052, 192, 384);
  clips.push_back(areaAbierta);
  /*
  //Abriendo
  for (int i = 0; i < 10; ++i) {
    Area area(1 + 194*i, 2052, 192, 384);
    clips.push_back(area);
  }
  for (int i = 0; i < 9; ++i) {
    Area area(1 + 194*i, 2437, 192, 384);
    clips.push_back(area);
  }

  //Cerrando
  for (int i = 0; i < 9; ++i) {
    Area area(1553 - i*194, 2437, 192, 384);
    clips.push_back(area);
  }
  for (int i = 0; i < 10; ++i) {
    Area area(1747 - 194*i, 2052, 192, 384);
    clips.push_back(area);
  }
  */
  //Cerrada
  Area areaCerrada(1553, 2437, 192, 384);
  clips.push_back(areaCerrada);
  tamanioHorizontal = 100;
  tamanioVertical = 200;

}

void VistaPuerta::dibujarEn(int x, int y) {
  Area srcArea = clips.at(estado);
  Area destArea(xInicial + x - tamanioHorizontal/2,
          yInicial + y - tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea);
}


void VistaPuerta::asignarEstado(int estado) {
  if (estado ≡ ABIERTA) this→estado = 1;
  else if (estado ≡ CERRADA) this→estado = 0;
}
```

jun 11, 19 16:12 **VistaPortalNaranja.h** Page 1/1

```
1  #ifndef VISTA_PORTAL_NARANJA
2  #define VISTA_PORTAL_NARANJA
3
4  #include "VistaObjeto.h"
5
6  class VistaPortalNaranja : public VistaObjeto {
7  private:
8    int angulo;
9  public:
10   VistaPortalNaranja(SdlTexture& tex, int angulo);
11   virtual void dibujarEn(int x, int y);
12   virtual ~VistaPortalNaranja() {}
13 };
14
15 #endif
```

jun 11, 19 16:12 **VistaPortalNaranja.cpp** Page 1/1

```
1  #include "VistaPortalNaranja.h"
2
3  VistaPortalNaranja::VistaPortalNaranja(SdlTexture& tex, int angulo) {
4    this→textura = tex;
5    this→angulo = angulo;
6    Area srcArea(385, 0, 30, 200);
7    clips.push_back(srcArea);
8    tamanioHorizontal = 15;
9    tamanioVertical = 100;
10 }
11
12 void VistaPortalNaranja::dibujarEn(int x, int y) {
13   Area srcArea = clips.front();
14   Area destArea(xInicial + x − tamanioHorizontal/2,
15           yInicial + y − tamanioVertical/2,
16           tamanioHorizontal, tamanioVertical);
17   textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
18 }
19
```

jun 11, 19 16:12 **VistaPortalAzul.h** Page 1/1

```
1  #ifndef VISTA_PORTAL_AZUL
2  #define VISTA_PORTAL_AZUL
3
4  #include "VistaObjeto.h"
5
6  class VistaPortalAzul : public VistaObjeto {
7  private:
8    int angulo;
9  public:
10   VistaPortalAzul(SdlTexture& tex, int angulo);
11   virtual void dibujarEn(int x, int y);
12   virtual ~VistaPortalAzul() {}
13 };
14
15 #endif
```

jun 11, 19 16:12 **VistaPortalAzul.cpp** Page 1/1

```
1  #include "VistaPortalAzul.h"
2
3  VistaPortalAzul::VistaPortalAzul(SdlTexture& tex, int angulo) {
4    this→textura = tex;
5    this→angulo = angulo;
6    Area srcArea(85, 0, 30, 200);
7    clips.push_back(srcArea);
8    tamanioHorizontal = 15;
9    tamanioVertical = 100;
10 }
11
12 void VistaPortalAzul::dibujarEn(int x, int y) {
13   Area srcArea = clips.front();
14   Area destArea(xInicial + x − tamanioHorizontal/2,
15           yInicial + y − tamanioVertical/2,
16           tamanioHorizontal, tamanioVertical);
17   textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
18 }
19
```

jun 11, 19 16:12 **VistaPinTool.h** Page 1/1

```
#ifndef VISTA_PIN_TOOL
#define VISTA_PIN_TOOL

#include "VistaObjeto.h"

class VistaPinTool : public VistaObjeto {
private:
  int color;
public:
  VistaPinTool(SdlTexture& tex);
  virtual void dibujarEn(int x, int y);
  virtual void asignarColorSegunPlayer(int playerId) override;
  virtual ~VistaPinTool() {}
};

#endif
```

jun 11, 19 16:12 **VistaPinTool.cpp** Page 1/1

```
#include "VistaPinTool.h"

#define CANT_CLIPS 6

VistaPinTool::VistaPinTool(SdlTexture& tex) {
  this→textura = tex;
  this→frame = 0;
  color = 0;
  for (int i = 0; i < CANT_CLIPS; ++i) {
    Area area(i*300, 50 + color*300, 300, 450);
    clips.push_back(area);
  }
  tamanioHorizontal = 30;
  tamanioVertical = 45;
}

void VistaPinTool::dibujarEn(int x, int y) {
  Area srcArea = clips.at(floor(frame/8));
  Area destArea(xInicial + x − tamanioHorizontal/2,
          yInicial + y − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea);
  ++frame;
  if (frame/8 ≥ CANT_CLIPS) {
    frame = 0;
  }
}

void VistaPinTool::asignarColorSegunPlayer(int playerId) {
  color = playerId − 1;
}
```

jun 11, 19 16:12 **VistaPiedraMovil.h** Page 1/1

```
#ifndef VISTA_PIEDRA_MOVIL
#define VISTA_PIEDRA_MOVIL

#include "VistaObjetoMovil.h"

class VistaPiedraMovil : public VistaObjetoMovil {
public:
  VistaPiedraMovil(SdlTexture& tex);
  void dibujarEn(int x, int y);
  void mover(int x, int y);
  virtual ~VistaPiedraMovil() {}
};

#endif
```

jun 11, 19 16:12 **VistaPiedraMovil.cpp** Page 1/1

```
#include "VistaPiedraMovil.h"

#include <stdlib.h>
#include <time.h>

VistaPiedraMovil::VistaPiedraMovil(SdlTexture& tex) {
  this→textura = tex;
  srand(time(NULL));
  posX = 0;
  posY = 0;
  int nroPiedra = rand() % 8;
  Area srcArea(1 + nroPiedra*86, 4513, 85, 83);
  clips.push_back(srcArea);
  tamanioHorizontal = 100;
  tamanioVertical = 100;
}

void VistaPiedraMovil::dibujarEn(int x, int y) {
  Area srcArea = clips.front();
  Area destArea(xInicial + x + posX − tamanioHorizontal/2,
          yInicial + y + posY − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea);
}

void VistaPiedraMovil::mover(int x, int y) {
  posX += x;
  posY += y;
}
```

jun 11, 19 16:12 **VistaPersonaje.h** Page 1/1

```
#ifndef VISTA_PERSONAJE
#define VISTA_PERSONAJE

#include "VistaObjetoMovil.h"
#include "AnimacionPersonaje.h"

class VistaPersonaje : public VistaObjetoMovil {
private:
  AnimacionPersonaje animaciones;
  int estado;
  SDL_RendererFlip rendererFlip;
public:
  VistaPersonaje(SdlTexture& tex);
  void dibujarEn(int x, int y);
  virtual void mover(int x, int y) override;
  virtual void asignarEstado(int estado) override;
  virtual void flip(int flip) override;
  virtual ~VistaPersonaje() {}
};

#endif
```

jun 11, 19 16:12 **VistaPersonaje.cpp** Page 1/1

```
#include "VistaPersonaje.h"
#include "../Common/Constantes.h"
#include <iostream>

VistaPersonaje::VistaPersonaje(SdlTexture& tex) {
  this→textura = tex;
  frame = 0;
  posX = 0;
  posY = 0;
  tamanioVertical = 105;
  estado = 0;
  rendererFlip = SDL_FLIP_NONE;
}

void VistaPersonaje::dibujarEn(int x, int y) {
  Area srcArea = animaciones.obtenerConEstado(estado, floor(frame/4));
  tamanioHorizontal = int(srcArea.getWidth()/2);
  Area destArea(xInicial + x + posX − tamanioHorizontal/2,
          yInicial + y + posY − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea, 0.0, rendererFlip);

  ++frame;
  if ((frame/4) ≥ animaciones.size()) {
    frame = 0;
  }
}

void VistaPersonaje::mover(int x, int y) {
  posX += x;
  posY += y;
}

void VistaPersonaje::asignarEstado(int estado) {
  if (estado ≠ this→estado) {
    frame = 0;
  }
  this→estado = estado;
}

void VistaPersonaje::flip(int flip) {
  if (flip ≡ DERECHA) rendererFlip = SDL_FLIP_NONE;
  else if (flip ≡ IZQUIERDA) rendererFlip = SDL_FLIP_HORIZONTAL;
}
```

jun 11, 19 16:12 **VistaObjetoMovil.h** Page 1/1

```
1  #ifndef VISTA_OBJETO_MOVIL
2  #define VISTA_OBJETO_MOVIL
3
4  #include "VistaObjeto.h"
5
6  class VistaObjetoMovil : public VistaObjeto {
7  protected:
8    int posX, posY;
9  public:
10   virtual void dibujarEn(int x, int y) = 0;
11   virtual void mover(int x, int y) = 0;
12   virtual ~VistaObjetoMovil() {}
13 };
14
15 #endif
```

jun 11, 19 16:12 **VistaObjetoMovil.cpp** Page 1/1

```
1  #include "VistaObjetoMovil.h"
```

jun 11, 19 16:12 **VistaObjeto.h** Page 1/1

```
#ifndef VISTA_OBJETO
#define VISTA_OBJETO

#include "Area.h"
#include "SdlTexture.h"
#include <vector>

class VistaObjeto {
protected:
  int id;
  SdlTexture textura;
  std::vector<Area> clips;
  int xInicial, yInicial;
  int frame;
  int tamanioHorizontal;
  int tamanioVertical;
public:
  virtual void dibujarEn(int x, int y) = 0;
  virtual void asignarEstado(int estado);
  virtual void mover(int x, int y);
  virtual void asignarRotacion(int angulo);
  virtual void flip(int flip);
  virtual void asignarColorSegunPlayer(int playerId);
  void setPosInicial(int x, int y);
  void setId(int id);
  int getId();
  virtual ~VistaObjeto() {}
};

#endif
```

jun 11, 19 16:12 **VistaObjeto.cpp** Page 1/1

```
#include "VistaObjeto.h"
#include <iostream>

void VistaObjeto::setPosInicial(int x, int y) {
  this→xInicial = x; this→yInicial = y;
}

void VistaObjeto::asignarEstado(int estado) {
  throw std::runtime_error("Asignando estado a objeto incorrecto");
}

void VistaObjeto::mover(int x, int y) {
  throw std::runtime_error("Intentando mover objeto incorrecto");
}

void VistaObjeto::flip(int flip) {
  throw std::runtime_error("Intentando flipear objeto incorrecto");
}

void VistaObjeto::asignarRotacion(int angulo) {
  throw std::runtime_error("Intentando rotar objeto incorrecto");
}

void VistaObjeto::asignarColorSegunPlayer(int playerId) {
  throw std::runtime_error("Intentando asignar color a objeto incorrecto");
}

void VistaObjeto::setId(int id) {
  this→id = id;
}

int VistaObjeto::getId() {
  return id;
}
```

jun 11, 19 16:12 **VistaFondo.h** Page 1/1

```
#ifndef VISTA_FONDO_H
#define VISTA_FONDO_H

#include "VistaObjeto.h"

class VistaFondo : public VistaObjeto {
public:
  VistaFondo(const SdlTexture& tex);
  void setDimensiones(int x, int y);
  virtual void dibujarEn(int x, int y) override;
  virtual ~VistaFondo() {}
};

#endif // VISTA_FONDO_H
```

jun 11, 19 16:12 **VistaFondo.cpp** Page 1/1

```
#include "VistaFondo.h"

#include <iostream>

VistaFondo::VistaFondo(const SdlTexture& tex) {
  this→textura = tex;
  Area srcArea(0, 0, 1680, 1050);
  clips.push_back(srcArea);
  tamanioHorizontal = 0;
  tamanioVertical = 0;
}

void VistaFondo::setDimensiones(int x, int y) {
  tamanioHorizontal = x;
  tamanioVertical = y;
}

void VistaFondo::dibujarEn(int x, int y) {
  Area srcArea = clips.front();
  Area destArea(x, y, tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea);
}

```

jun 11, 19 16:12 **VistaEmisor.h** Page 1/1

```
1  #ifndef VISTA_EMISOR
2  #define VISTA_EMISOR
3
4  #include "VistaObjeto.h"
5
6  class VistaEmisor : public VistaObjeto {
7  private:
8    int angulo;
9  public:
10   VistaEmisor(SdlTexture& tex, int angulo);
11   virtual void dibujarEn(int x, int y);
12   virtual ~VistaEmisor() {}
13 };
14
15 #endif
```

jun 11, 19 16:12 **VistaEmisor.cpp** Page 1/1

```
1  #include "VistaEmisor.h"
2
3  VistaEmisor::VistaEmisor(SdlTexture& tex, int angulo) {
4    this→textura = tex;
5    this→angulo = angulo;
6    Area srcArea(1, 0, 191, 191);
7    clips.push_back(srcArea);
8    tamanioHorizontal = 100;
9    tamanioVertical = 100;
10 }
11
12 void VistaEmisor::dibujarEn(int x, int y) {
13   Area srcArea = clips.front();
14   Area destArea(xInicial + x − tamanioHorizontal/2,
15           yInicial + y − tamanioVertical/2,
16           tamanioHorizontal, tamanioVertical);
17   textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
18 }
```

jun 11, 19 16:12 **VistaBoton.h** Page 1/1

```
1  #ifndef VISTA_BOTON
2  #define VISTA_BOTON
3
4  #include "VistaObjeto.h"
5
6  class VistaBoton : public VistaObjeto {
7  private:
8    int estado;
9  public:
10   VistaBoton(SdlTexture& tex);
11   virtual void asignarEstado(int estado) override;
12   virtual void dibujarEn(int x, int y);
13   virtual ~VistaBoton() {}
14 };
15
16 #endif
```

jun 11, 19 16:12 **VistaBoton.cpp** Page 1/1

```
1  #include "VistaBoton.h"
2  #include "../Common/Constantes.h"
3
4  #define CANT_CLIPS 2
5
6  VistaBoton::VistaBoton(SdlTexture& tex) {
7    this→textura = tex;
8    this→estado = 0;
9    for (int i = 0; i < CANT_CLIPS; ++i) {
10     Area area(1, 116 + i*76, 175, 55);
11     clips.push_back(area);
12   }
13   tamanioHorizontal = 100;
14   tamanioVertical = 30;
15
16 }
17
18 void VistaBoton::dibujarEn(int x, int y) {
19   Area srcArea = clips.at(estado);
20   Area destArea(xInicial + x − tamanioHorizontal/2,
21           yInicial + y − tamanioVertical/2,
22           tamanioHorizontal, tamanioVertical);
23   textura.render(srcArea, destArea);
24 }
25
26 void VistaBoton::asignarEstado(int estado) {
27   if (estado ≡ PRESIONADO) this→estado = 1;
28   else if (estado ≡ NO_PRESIONADO) this→estado = 0;
29 }
```

jun 11, 19 16:12 **VistaBolaEnergia.h** Page 1/1

```
#ifndef VISTA_BOLA_ENERGIA
#define VISTA_BOLA_ENERGIA

#include "VistaObjetoMovil.h"

class VistaBolaEnergia : public VistaObjetoMovil {
private:
  int angulo;
public:
  VistaBolaEnergia(SdlTexture& tex, int angulo);
  virtual void asignarRotacion(int rotacion) override;
  void dibujarEn(int x, int y);
  void mover(int x, int y);
  virtual ~VistaBolaEnergia() {}
};

#endif
```

jun 11, 19 16:12 **VistaBolaEnergia.cpp** Page 1/1

```
#include "VistaBolaEnergia.h"

#define CANT_CLIPS 3

VistaBolaEnergia::VistaBolaEnergia(SdlTexture& tex, int angulo) {
  this→textura = tex;
  this→angulo = angulo;
  frame = 0;
  posX = 0;
  posY = 0;
  for (int i = 0; i < CANT_CLIPS; ++i) {
    Area area(i*115, 1922, 115, 60);
    clips.push_back(area);
  }
  tamanioHorizontal = 100;
  tamanioVertical = 45;
}

void VistaBolaEnergia::dibujarEn(int x, int y) {
  Area srcArea = clips.at(floor(frame/4));
  Area destArea(xInicial + x + posX − tamanioHorizontal/2,
          yInicial + y + posY − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
  ++frame;
  if ((frame/4) ≥ CANT_CLIPS) {
    frame = 0;
  }
}

void VistaBolaEnergia::mover(int x, int y) {
  posX += x;
  posY += y;
}

void VistaBolaEnergia::asignarRotacion(int rotacion) {
  angulo = rotacion;
}
```

jun 11, 19 16:12 **VistaBloquePiedra.h** Page 1/1

```
1  #ifndef VISTA_BLOQUE_PIEDRA
2  #define VISTA_BLOQUE_PIEDRA
3
4  #include "VistaObjeto.h"
5
6  class VistaBloquePiedra : public VistaObjeto {
7  public:
8    VistaBloquePiedra(const SdlTexture& tex);
9    void dibujarEn(int x, int y) override;
10   virtual ~VistaBloquePiedra() {}
11 };
12
13 #endif
```

jun 11, 19 16:12 **VistaBloquePiedra.cpp** Page 1/1

```
1  #include "VistaBloquePiedra.h"
2  #include <iostream>
3
4  VistaBloquePiedra::VistaBloquePiedra(const SdlTexture& tex) {
5    this→textura = tex;
6    Area srcArea(2, 0, 191, 191);
7    clips.push_back(srcArea);
8    tamanioHorizontal = 100;
9    tamanioVertical = 100;
10 }
11
12 void VistaBloquePiedra::dibujarEn(int x, int y) {
13   Area srcArea = clips.front();
14   Area destArea(xInicial + x − tamanioHorizontal/2,
15           yInicial + y − tamanioVertical/2,
16           tamanioHorizontal, tamanioVertical);
17   textura.render(srcArea, destArea);
18 }
```

jun 11, 19 16:12 **VistaBloqueMetal.h** Page 1/1

```
#ifndef VISTA_BLOQUE_METAL
#define VISTA_BLOQUE_METAL

#include "VistaObjeto.h"

class VistaBloqueMetal : public VistaObjeto {
public:
  VistaBloqueMetal(SdlTexture& tex);
  void dibujarEn(int x, int y) override;
  virtual ~VistaBloqueMetal() {}
};

#endif
```

jun 11, 19 16:12 **VistaBloqueMetalDiagonal.h** Page 1/1

```
#ifndef VISTA_BLOQUE_METAL_DIAGONAL
#define VISTA_BLOQUE_METAL_DIAGONAL

#include "VistaObjeto.h"

class VistaBloqueMetalDiagonal : public VistaObjeto {
private:
  int angulo;
public:
  VistaBloqueMetalDiagonal(SdlTexture& tex, int angulo);
  virtual void dibujarEn(int x, int y);
  virtual ~VistaBloqueMetalDiagonal() {}
};

#endif
```

jun 11, 19 16:12 **VistaBloqueMetalDiagonal.cpp** Page 1/1

```cpp
#include "VistaBloqueMetalDiagonal.h"

VistaBloqueMetalDiagonal::VistaBloqueMetalDiagonal(SdlTexture& tex, int angulo)
{
  this→textura = tex;
  this→angulo = angulo;
  Area srcArea(2, 405, 185, 185);
  clips.push_back(srcArea);
  tamanioHorizontal = 100;
  tamanioVertical = 100;
}

void VistaBloqueMetalDiagonal::dibujarEn(int x, int y) {
  Area srcArea = clips.front();
  Area destArea(xInicial + x − tamanioHorizontal/2,
          yInicial + y − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
}
```

jun 11, 19 16:12 **VistaBloqueMetal.cpp** Page 1/1

```cpp
#include "VistaBloqueMetal.h"

VistaBloqueMetal::VistaBloqueMetal(SdlTexture& tex) {
  this→textura = tex;
  Area srcArea(2, 213, 191, 191);
  clips.push_back(srcArea);
  tamanioHorizontal = 100;
  tamanioVertical = 100;
}

void VistaBloqueMetal::dibujarEn(int x, int y) {
  Area srcArea = clips.front();
  Area destArea(xInicial + x − tamanioHorizontal/2,
          yInicial + y − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea);
}
```

jun 11, 19 16:12 **VistaBarreraEnergia.h** Page 1/1

```
#ifndef VISTA_BARRERA_ENERGIA
#define VISTA_BARRERA_ENERGIA

#include "VistaObjeto.h"

class VistaBarreraEnergia : public VistaObjeto {
private:
  int angulo;
public:
  VistaBarreraEnergia(SdlTexture& tex, int angulo);
  virtual void dibujarEn(int x, int y);
  virtual ~VistaBarreraEnergia() {}
};

#endif
```

jun 11, 19 16:12 **VistaBarreraEnergia.cpp** Page 1/1

```
#include "VistaBarreraEnergia.h"

VistaBarreraEnergia::VistaBarreraEnergia(SdlTexture& tex, int angulo) {
  this→textura = tex;
  this→angulo = angulo;
  Area srcArea(1, 1722, 225, 50);
  clips.push_back(srcArea);
  tamanioHorizontal = 100;
  tamanioVertical = 50;
}

void VistaBarreraEnergia::dibujarEn(int x, int y) {
  Area srcArea = clips.front();
  Area destArea(xInicial + x − tamanioHorizontal/2,
          yInicial + y − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea, (double) angulo, SDL_FLIP_NONE);
}
```

jun 11, 19 16:12 **VistaAcido.h** Page 1/1

```
#ifndef VISTA_ACIDO
#define VISTA_ACIDO

#include "VistaObjeto.h"

class VistaAcido : public VistaObjeto {
public:
  VistaAcido(SdlTexture& tex);
  virtual void dibujarEn(int x, int y);
  virtual ~VistaAcido() {}
};

#endif
```

jun 11, 19 16:12 **VistaAcido.cpp** Page 1/1

```
#include "VistaAcido.h"

#define CANT_CLIPS 8

VistaAcido::VistaAcido(SdlTexture& tex) {
  this→textura = tex;
  frame = 0;
  for (int i = 0; i < CANT_CLIPS; ++i) {
    Area area(i*301, 1553, 301, 60);
    clips.push_back(area);
  }
  tamanioHorizontal = 100;
  tamanioVertical = 20;
}

void VistaAcido::dibujarEn(int x, int y) {
  Area srcArea = clips.at(floor(frame/4));
  Area destArea(xInicial + x − tamanioHorizontal/2,
          yInicial + y − tamanioVertical/2,
          tamanioHorizontal, tamanioVertical);
  textura.render(srcArea, destArea);
  ++frame;
  if (frame/4 ≥ CANT_CLIPS) {
    frame = 0;
  }
}
```

jun 11, 19 16:12 **SeleccionadorEscena.h** Page 1/1

```h
#ifndef SELECCIONADOR_ESCENA_H
#define SELECCIONADOR_ESCENA_H

#include "SdlWindow.h"
#include "../Common/Cola.h"
#include "../Common/cola_bloqueante.h"
#include "EnviadorEventos.h"
#include "RecibidorEventos.h"
#include "../Common/Evento.h"
#include <map>

// Forward Declaration
class EscenaBase;

class SeleccionadorEscena {
private:
  SdlWindow window;
  Socket socket;

  ColaBloqueante<Evento*> colaEnviar;
  Cola<Evento*> colaRecibir;

  RecibidorEventos recibidorEventos;
  EnviadorEventos enviadorEventos;

  std::map<int, EscenaBase*> escenas;
  int escenaActual;
public:
  SeleccionadorEscena(int xScreen, int yScreen);
  void ejecutar();
  ~SeleccionadorEscena();
};

#endif // SELECCIONADOR_ESCENA_H
```

jun 11, 19 16:12 **SeleccionadorEscena.cpp** Page 1/1

```cpp
#include "SeleccionadorEscena.h"
#include "EscenaJuego.h"
#include "Menu.h"
#include "../Common/Constantes.h"
#include "Menu.h"

SeleccionadorEscena::SeleccionadorEscena(int xScreen, int yScreen) :
  window(xScreen, yScreen),
  recibidorEventos(colaRecibir, socket),
  enviadorEventos(colaEnviar, socket),
  escenaActual(ESCENA_MENU) {

  socket.conectar("localhost", "8888");

  Evento* eventoIniciar = new EventoIniciarPartida();
  colaEnviar.put(eventoIniciar);

  recibidorEventos.iniciar();
  enviadorEventos.iniciar();

  escenas[ESCENA_JUEGO] = new EscenaJuego(window, colaEnviar, colaRecibir);
  escenas[ESCENA_MENU] = new Menu(window);
}

void SeleccionadorEscena::ejecutar() {
  EscenaBase* escena;
  while(true) {
      escena = escenas.at(escenaActual);
    escena→actualizar();
    escena→dibujar();
    escenaActual = escena→manejarEventos();
    if (escena→termino()) break;
  }
}

SeleccionadorEscena::~SeleccionadorEscena() {
  recibidorEventos.detener();
  enviadorEventos.detener();
  for (auto& it : escenas) {
    delete it.second;
  }
}
```

jun 11, 19 16:12 **SdlWindow.h** Page 1/1

```
#ifndef __SDL_WINDOW_H__
#define __SDL_WINDOW_H__
#include "Area.h"

class SDL_Window;
class SDL_Renderer;
class SDL_Texture;

class SdlWindow {
public:
    /**
     * Ctor standalone
     */
    SdlWindow(int width, int height);
    ~SdlWindow();
    void fill();
    void fill(int r, int g, int b, int alpha);
    void render();
    void setFullscreen(bool fullscreen);
    void getWindowSize(int* x, int* y);
    SDL_Renderer* getRenderer() const;
private:
    int width;
    int height;
    SDL_Window* window;
    SDL_Renderer* renderer;
};

#endif
```

jun 11, 19 16:12 **SdlWindow.cpp** Page 1/1

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_video.h>
#include <SDL2/SDL_render.h>
#include "SdlException.h"
#include "SdlWindow.h"
#include <iostream>


SdlWindow::SdlWindow(int width, int height) :
        width(width), height(height) {
    int errCode = SDL_Init(SDL_INIT_VIDEO);
    if (errCode) {
        throw SdlException("Error en la inicializaciÃ³n", SDL_GetError());
    }
    errCode = SDL_CreateWindowAndRenderer(
        width, height, SDL_RENDERER_ACCELERATED,
        &this→window, &this→renderer);
    if (errCode) {
        throw SdlException("Error al crear ventana", SDL_GetError());
    }

    SDL_SetWindowFullscreen(window, SDL_WINDOW_FULLSCREEN_DESKTOP);
}


SdlWindow::~SdlWindow() {
    std::cout << "Destruyendo" << std::endl;
    if (this→renderer) {
        SDL_DestroyRenderer(this→renderer);
        this→renderer = nullptr;
    }

    if (this→window) {
        SDL_DestroyWindow(this→window);
        this→window = nullptr;
    }
}

void SdlWindow::fill(int r, int g, int b, int alpha) {
    SDL_SetRenderDrawColor(this→renderer,
                           r, g, b, alpha);
    SDL_RenderClear(this→renderer);
}

void SdlWindow::fill() {
    this→fill(0x33,0x33,0x33,0xFF);
}

void SdlWindow::render() {
    SDL_RenderPresent(this→renderer);
}


SDL_Renderer* SdlWindow::getRenderer() const {
    return this→renderer;
}

void SdlWindow::setFullscreen(bool fullscreen) {
    if (fullscreen) SDL_SetWindowFullscreen(window, SDL_WINDOW_FULLSCREEN_DESKTO
P);
    else SDL_SetWindowFullscreen(window, 0);
}

void SdlWindow::getWindowSize(int* x, int* y) {
    SDL_GetWindowSize(window, x, y);
}
```

jun 11, 19 16:12 **SdlTexture.h** Page 1/1

```
#ifndef __SDL_TEXTURE_H__
#define __SDL_TEXTURE_H__
#include <string>
#include <cstdint>
#include <SDL2/SDL.h>


class SDL_Texture;
class SDL_Renderer;
class SdlWindow;
class Area;

class SdlTexture {
public:
    SdlTexture();
    /**
     * Crea un SDL_Texture, lanza una excepciÃ³n si el filename es invÃ¡lido
     **/
    SdlTexture(const std::string &filename, const SdlWindow& window);
    /**
     * Libera la memoria reservada por la textura
     **/
    ~SdlTexture();
    /**
     * Renderiza la textura cargada
     **/
    int render(const Area& src, const Area& dest) const;
    int render(const Area& src, const Area& dest, double grados, SDL_RendererFli
p flip) const;
    /**
     * Asigna una opacidad a la textura
     **/
    void setOpacity(const float opacity);

    //Asigna un color para agregar a la textura
    void setColor(int r, int g, int b);
private:
    SDL_Texture* loadTexture(const std::string &filename);
    SDL_Renderer* renderer;
    SDL_Texture* texture;
};

#endif
```

jun 11, 19 16:12 **SdlTexture.cpp** Page 1/1

```
#include "SdlTexture.h"
#include <SDL2/SDL_image.h>
#include <string>
#include "SdlWindow.h"
#include "SdlException.h"

SdlTexture::SdlTexture() {}

SdlTexture::SdlTexture(const std::string &filename, const SdlWindow& window)
    : renderer(window.getRenderer()) {
    this→texture = loadTexture(filename);
}

SdlTexture::~SdlTexture() {
    SDL_DestroyTexture(this→texture);
}

SDL_Texture* SdlTexture::loadTexture(const std::string &filename) {
    SDL_Texture* texture = IMG_LoadTexture(this→renderer,
                                           filename.c_str());
    if (¬texture) {
        throw SdlException("Error al cargar la textura", SDL_GetError());
    }
    return texture;
}

int SdlTexture::render(const Area& src, const Area& dest) const {
    SDL_Rect sdlSrc = {
            src.getX(), src.getY(),
            src.getWidth(), src.getHeight()
    };
    SDL_Rect sdlDest = {
            dest.getX(), dest.getY(),
            dest.getWidth(), dest.getHeight()
    };
    return SDL_RenderCopy(this→renderer, this→texture, &sdlSrc, &sdlDest);
}

int SdlTexture::render(const Area& src, const Area& dest, double grados, SDL_Ren
dererFlip flip) const {
    SDL_Rect sdlSrc = {
            src.getX(), src.getY(),
            src.getWidth(), src.getHeight()
    };
    SDL_Rect sdlDest = {
            dest.getX(), dest.getY(),
            dest.getWidth(), dest.getHeight()
    };
    return SDL_RenderCopyEx(this→renderer, this→texture, &sdlSrc, &sdlDest, gr
ados, NULL, flip);
}

void SdlTexture::setOpacity(const float opacity) {
    SDL_SetTextureAlphaMod(this→texture, opacity);
}

void SdlTexture::setColor(int r, int g, int b) {
    SDL_SetTextureColorMod(this→texture, r, g, b);
}
```

jun 11, 19 16:12 **SdlException.h** Page 1/1

```h
#ifndef __SDL_EXCEPTION_H__
#define __SDL_EXCEPTION_H__
#include <string>
#include <exception>

class SdlException : public std::exception {
public:
    SdlException(const char* description, const char* sdlError);
    const char* what() const noexcept;
private:
    std::string description;
};

#endif
```

jun 11, 19 16:12 **SdlException.cpp** Page 1/1

```cpp
#include "SdlException.h"
#include <string>
#include <SDL2/SDL_image.h>

SdlException::SdlException(const char* description, const char* sdlError)
    : std::exception(), description(description) {
    this→description.append("\n SDL_ERROR: ").append(sdlError);
}

const char* SdlException::what() const noexcept{
    return this→description.c_str();
}
```

jun 11, 19 16:12 **RecibidorEventos.h** Page 1/1

```
1  #ifndef RECIBIR_EVENTOS
2  #define RECIBIR_EVENTOS
3
4  #include "../Common/Thread.h"
5  #include "../Common/Cola.h"
6  #include "../Common/Evento.h"
7  #include "../Common/Serializador.h"
8  #include "../Common/Socket.h"
9
10 class RecibidorEventos : public Thread {
11 private:
12   bool termino;
13   Cola<Evento*>& cola;
14   Serializador serializador;
15   Socket& socket;
16 public:
17   RecibidorEventos(Cola<Evento*>& cola, Socket& socket);
18   virtual void ejecutar();
19   void detener();
20 };
21
22 #endif
```

jun 11, 19 16:12 **RecibidorEventos.cpp** Page 1/1

```
1  #include "RecibidorEventos.h"
2
3  #include <iostream>
4
5  RecibidorEventos::RecibidorEventos(Cola<Evento*>& cola, Socket& socket) :
6    cola(cola), socket(socket) {
7    termino = false;
8    }
9
10 void RecibidorEventos::ejecutar() {
11   while(¬termino) {
12     Evento* evento = serializador.recibirEvento(socket);
13     cola.put(evento);
14   }
15 }
16
17 void RecibidorEventos::detener() {
18   termino = true;
19   socket.shutdown();
20   Thread::cerrar();
21 }
```

jun 11, 19 16:12 **Menu.h** Page 1/1

```
1  #ifndef MENU_H
2  #define MENU_H
3
4  #include "SdlWindow.h"
5  #include "SdlTexture.h"
6  #include "VistaFondo.h"
7  #include "Audio.h"
8  #include "../Common/Socket.h"
9  #include "EscenaBase.h"
10
11 class Menu : public EscenaBase {
12 private:
13   int xScreen;
14   int yScreen;
15   SdlWindow& window;
16   SdlTexture imagenMenuTex;
17   VistaFondo fondo;
18   Audio audio;
19
20   SDL_Event e;
21   bool terminado;
22   int siguienteEscena;
23
24   Area botonJugar;
25   Area botonSalir;
26 public:
27   Menu(SdlWindow& window);
28   virtual bool termino() override;
29   virtual void actualizar() override;
30   virtual void dibujar() override;
31   virtual int manejarEventos() override;
32   virtual void manejar(Evento& evento) {};
33   virtual ~Menu() {}
34 };
35
36 #endif // MENU_H
```

jun 11, 19 16:12 **Menu.cpp** Page 1/2

```
1  #include "Menu.h"
2  #include "EnviadorEventos.h"
3  #include "RecibidorEventos.h"
4  #include "SdlWindow.h"
5  #include "../Common/Evento.h"
6  #include "../Common/Constantes.h"
7  #include <thread>
8  #include <chrono>
9  #include <iostream>
10
11 Menu::Menu(SdlWindow& window) :
12   window(window),
13   imagenMenuTex("menu.png", window),
14   fondo(imagenMenuTex) {
15     window.setFullscreen(true);
16     terminado = false;
17     audio.reproducirMusica();
18   }
19
20 bool Menu::termino() {
21   return terminado;
22 }
23
24 void Menu::actualizar() {
25   //Poner los botones en posiciones relativas al tamaÃ±o total de la pantalla
26   int xJugar = 0.75*(xScreen/2);
27   int yJugar = 0.6*(yScreen);
28   int wJugar = 0.5*(xScreen/2);
29   int hJugar = 0.12*(yScreen);
30   botonJugar.set(xJugar, yJugar, wJugar, hJugar);
31
32   int xSalir = 0.79*(xScreen/2);
33   int ySalir = 0.74*(yScreen);
34   int wSalir = 0.2*(xScreen);
35   int hSalir = 0.1*(yScreen);
36   botonSalir.set(xSalir, ySalir, wSalir, hSalir);
37 }
38
39 void Menu::dibujar() {
40   window.fill();
41   window.getWindowSize(&xScreen, &yScreen);
42   fondo.setDimensiones(xScreen, yScreen);
43   fondo.dibujarEn(0, 0);
44   std::this_thread::sleep_for(std::chrono::milliseconds(1));
45   window.render();
46 }
47
48 int Menu::manejarEventos() {
49   siguienteEscena = ESCENA_MENU;
50   while (SDL_PollEvent(&e) ∧ ¬terminado) {
51     if (e.type ≡ SDL_QUIT) {
52       terminado = true;
53     } else if (e.type ≡ SDL_MOUSEBUTTONDOWN) {
54       int x, y;
55       SDL_GetMouseState(&x, &y);
56       if (botonSalir.estaAdentro(x, y)) {
57         terminado = true;
58       }
59       if (botonJugar.estaAdentro(x, y)) {
60         audio.pararMusica();
61         siguienteEscena = ESCENA_JUEGO;
62       }
63     } else if (e.type ≡ SDL_KEYDOWN) {
64       SDL_KeyboardEvent& keyEvent = (SDL_KeyboardEvent&) e;
65       if (keyEvent.keysym.sym ≡ SDLK_F11) window.setFullscreen(false);
66     }
```

jun 11, 19 16:12 **Menu.cpp** Page 2/2

```
67      }
68      return siguienteEscena;
69    }
70
71
```

jun 11, 19 16:12 **mainCliente.cpp** Page 1/1

```
1   #include "SeleccionadorEscena.h"
2   #include <iostream>
3
4   int main(int argc, char** argv) {
5     try {
6       SeleccionadorEscena seleccionadorEscena(1000, 800);
7       seleccionadorEscena.ejecutar();
8     } catch (std::runtime_error& e) {
9       std::cout << e.what() << std::endl;
10    }
11    return 0;
12  }
```

jun 11, 19 16:12 **InputHandler.h** Page 1/1

```
#ifndef INPUT_HANDLER_H
#define INPUT_HANDLER_H

#include <SDL2/SDL.h>
#include "SdlWindow.h"
#include "../Common/Constantes.h"
#include "../Common/cola_bloqueante.h"
#include "Audio.h"
#include "../Common/Conversor.h"

class Evento;

class InputHandler {
private:
  SdlWindow& window;
  ColaBloqueante<Evento*>& colaEnviar;
  Audio& audio;
  Conversor conv;

  SDL_Event event;
  int playerId;
  bool terminado;
  bool ctrl;
  bool fullscreen;
public:
  InputHandler(SdlWindow& window, ColaBloqueante<Evento*>& colaEnviar, Audio& audio);
  void setPlayerId(int playerId);
  bool termino();
  void handle();
};

#endif // INPUT_HANDLER_H
```

jun 11, 19 16:12 **InputHandler.cpp** Page 1/2

```
#include "InputHandler.h"
#include <iostream>
#include "../Common/Evento.h"

InputHandler::InputHandler(SdlWindow& window, ColaBloqueante<Evento*>& colaEnviar, Audio& audio) :
  window(window),
  colaEnviar(colaEnviar),
  audio(audio),
  conv(100) {
  terminado = false;
  fullscreen = true;
  ctrl = false;
}

void InputHandler::setPlayerId(int playerId) {
  this→playerId = playerId;
}

bool InputHandler::termino() {
  return terminado;
}

void InputHandler::handle() {
  Evento* evento;
  while (SDL_PollEvent(&event)) {
    if (event.type ≡ SDL_QUIT) {
      terminado = true;
    } else if (event.type ≡ SDL_MOUSEBUTTONDOWN) {
      int x, y, xScreen, yScreen;
      SDL_GetMouseState(&x, &y);
      window.getWindowSize(&xScreen, &yScreen);
      //int dirX = x − (xScreen/2);
      //int dirY = y − (yScreen/2 + 50);
      if (ctrl) {
        //evento = new EventoPinTool(conv.pixelABloque(x), conv.pixelABloque(y));
        //colaEnviar.put(evento);
        evento = new EventoCrearItem(ID_PIN_TOOL, x, y, 0);
        colaEnviar.put(evento);
      }
      else if (event.button.button ≡ SDL_BUTTON_LEFT) {
        //evento = new EventoPortalAzul(dirX, dirY);
        //colaEnviar.put(evento);
        evento = new EventoCambioEstado(ESTADO_DISPARANDO, playerId);
        colaEnviar.put(evento);
        evento = new EventoCrearItem(ID_PORTAL_AZUL, x, y, 90);
        colaEnviar.put(evento);
        audio.reproducirEfecto(EFECTO_DISPARO);
      }
      else if (event.button.button ≡ SDL_BUTTON_RIGHT) {
        //evento = new EventoPortalNaranja(dirX, dirY);
        //colaEnviar.put(evento);
        evento = new EventoCambioEstado(ESTADO_DISPARANDO, playerId);
        colaEnviar.put(evento);
        evento = new EventoCrearItem(ID_PORTAL_NARANJA, x, y, 0);
        colaEnviar.put(evento);
        audio.reproducirEfecto(EFECTO_DISPARO);
      }

    } else if (event.type ≡ SDL_KEYDOWN) {
      SDL_KeyboardEvent& keyEvent = (SDL_KeyboardEvent&) event;
      switch (keyEvent.keysym.sym) {
        case SDLK_a: {
          //evento = new EventoCorrer(IZQUIERDA, playerId);
          //colaEnviar.put(evento); mando esto en realidad, pero simulo lo que m
```

```
    e manda el server con lo de abajo
65            evento = new EventoMover(-15, 0, playerId);
66            colaEnviar.put(evento);
67            evento = new EventoCambioEstado(ESTADO_CORRIENDO, playerId);
68            colaEnviar.put(evento);
69            evento = new EventoFlip(IZQUIERDA, playerId);
70            colaEnviar.put(evento);
71            break;
72          }
73          case SDLK_d: {
74            // evento = new EventoCorrer(DERECHA, playerId);
75            //colaEnviar.put(evento); mando esto en realidad, pero simulo lo que m
    e manda el server con lo de abajo
76            evento = new EventoMover(15, 0, playerId);
77            colaEnviar.put(evento);
78            evento = new EventoCambioEstado(ESTADO_CORRIENDO, playerId);
79            colaEnviar.put(evento);
80            evento = new EventoFlip(DERECHA, playerId);
81            colaEnviar.put(evento);
82            break;
83          }
84          case SDLK_w: {
85            // evento = new EventoSalto(playerId);
86            //colaEnviar.put(evento); mando esto en realidad, pero simulo lo que m
    e manda el server con lo de abajo
87            evento = new EventoMover(0, -10, playerId);
88            colaEnviar.put(evento);
89            evento = new EventoCambioEstado(ESTADO_SALTANDO, playerId);
90            colaEnviar.put(evento);
91            audio.reproducirEfecto(EFECTO_SALTO);
92            break;
93          }
94          case SDLK_k: {
95            //evento = new EventoSuicidio(playerId);
96            //colaEnviar.put(evento);
97            evento = new EventoCambioEstado(ESTADO_MUERTO, playerId);
98            colaEnviar.put(evento);
99            break;
100         }
101         case SDLK_F11:
102           if (fullscreen) {
103             window.setFullscreen(false);
104             fullscreen = false;
105           } else {
106             window.setFullscreen(true);
107             fullscreen = true;
108           }
109           break;
110         case SDLK_LCTRL: ctrl = true; break;
111       }
112
113     } else if (event.type == SDL_KEYUP) {
114       SDL_KeyboardEvent& keyEvent = (SDL_KeyboardEvent&) event;
115       if (keyEvent.keysym.sym == SDLK_LCTRL) {
116         ctrl = false;
117       } else {
118         //evento = new EventoDejarDeMoverse(playerId);
119         //colaEnviar.put(evento);
120         evento = new EventoCambioEstado(ESTADO_IDLE, playerId);
121         colaEnviar.put(evento);
122       }
123     }
124   }
125 }
```



```
1  #ifndef ESCENA_JUEGO_H
2  #define ESCENA_JUEGO_H
3
4  #include "EscenaBase.h"
5  #include <SDL2/SDL.h>
6  #include <vector>
7  #include <memory>
8  #include "SdlWindow.h"
9  #include "SdlTexture.h"
10 #include "VistaFondo.h"
11 #include "CreadorTexturas.h"
12 #include "../Common/Cola.h"
13 #include "../Common/cola_bloqueante.h"
14 #include "../Common/Evento.h"
15 #include "Audio.h"
16 #include "../Common/Conversor.h"
17 #include "InputHandler.h"
18
19 typedef std::shared_ptr<VistaObjeto> VistaObjetoPtr;
20
21 //FD
22 class Evento;
23 class EventoCrearItem;
24 class EventoMover;
25 class EventoFlip;
26 class EventoCambioEstado;
27 class EventoEliminarItem;
28 class EventoRotacion;
29 class EventoCreacionPersonaje;
30
31 class EscenaJuego : public EscenaBase {
32 private:
33   Audio audio;
34   SdlWindow window;
35
36   Conversor conv;
37   CreadorTexturas creadorTexturas;
38   std::map<int, VistaObjetoPtr> objetosDelJuego;
39   SdlTexture fondoTex;
40   VistaFondo fondo;
41
42   ColaBloqueante<Evento*>& colaEnviar;
43   Cola<Evento*>& colaRecibir;
44
45   InputHandler handler;
46   bool terminado;
47
48   int miId;
49   int deltaCamaraX, deltaCamaraY;
50
51 public:
52   EscenaJuego(SdlWindow& window, ColaBloqueante<Evento*>& colaEnviar,
53     Cola<Evento*>& colaRecibir);
54   virtual bool termino() override;
55   virtual void actualizar() override;
56   virtual void dibujar() override;
57   virtual int manejarEventos() override;
58   virtual ~EscenaJuego();
59
60   virtual void manejar(Evento& evento) override;
61   virtual void manejar(EventoCrearItem& evento) override;
62   virtual void manejar(EventoMover& evento) override;
63   virtual void manejar(EventoFlip& evento) override;
64   virtual void manejar(EventoCambioEstado& evento) override;
65   virtual void manejar(EventoEliminarItem& evento) override;
66   virtual void manejar(EventoRotacion& evento) override;
```

jun 11, 19 16:12 **EscenaJuego.h** Page 2/2

```
67    virtual void manejar(EventoCreacionPersonaje& evento) override;
68  private:
69    void crearTerreno();
70    void recibirMiIdentificador();
71    void actualizarFondo();
72  };
73
74  #endif
```

jun 11, 19 16:12 **EscenaJuego.cpp** Page 1/3

```
1   #include "EscenaJuego.h"
2   #include <iostream>
3
4   #include <thread>
5   #include <chrono>
6
7   #include "yaml-cpp/yaml.h"
8   #include "../Common/Constantes.h"
9
10  EscenaJuego::EscenaJuego(SdlWindow& window, ColaBloqueante<Evento*>& colaEnviar,
     Cola<Evento*>& colaRecibir) :
11    window(window),
12    conv(100),
13    creadorTexturas(window),
14    fondoTex("fondo.png", window),
15    fondo(fondoTex),
16    colaEnviar(colaEnviar),
17    colaRecibir(colaRecibir),
18    handler(window, colaEnviar, audio) {
19
20    audio.reproducirMusica();
21    terminado = false;
22    miId = 0;
23    deltaCamaraX = 0;
24    deltaCamaraY = 0;
25    window.fill();
26    fondoTex.setColor(64, 64, 64);
27    recibirMiIdentificador();
28    crearTerreno();
29  }
30
31  bool EscenaJuego::termino() {
32    return terminado;
33  }
34
35  void EscenaJuego::actualizar() {
36    Evento* evento;
37    while (colaRecibir.get(evento)) {
38      evento->actualizar(*this);
39      delete evento;
40    }
41  }
42
43  void EscenaJuego::dibujar() {
44    window.fill();
45    std::this_thread::sleep_for(std::chrono::milliseconds(20));
46    actualizarFondo();
47    for (auto& it : objetosDelJuego) {
48      it.second->dibujarEn(deltaCamaraX, deltaCamaraY);
49    }
50    window.render();
51  }
52
53  int EscenaJuego::manejarEventos() {
54    handler.handle();
55    if (handler.termino()) {
56      terminado = true;
57    }
58    //Se retorna la escena siguiente, en el caso del juego
59    // es la escena final, por lo que se devuelve esta constante.
60    return ESCENA_JUEGO;
61  }
62
63  void EscenaJuego::manejar(Evento& evento) {
64    evento.actualizar(*this);
65  }
```

```
66
67  void EscenaJuego::manejar(EventoCrearItem& evento) {
68    VistaObjetoPtr vo = creadorTexturas.crear(
69                evento.atributos["idItem"],
70                evento.atributos["x"] - deltaCamaraX,
71                evento.atributos["y"] - deltaCamaraY,
72                evento.atributos["angulo"]);
73    objetosDelJuego.insert(std::make_pair(vo→getId(), vo));
74  }
75
76  void EscenaJuego::manejar(EventoMover& evento) {
77    objetosDelJuego.at(evento.atributos["idLanzador"])
78        →mover(evento.atributos["x"], evento.atributos["y"]);
79    if (evento.atributos["idLanzador"] ≡ miId) {
80      deltaCamaraX -= evento.atributos["x"];
81      deltaCamaraY -= evento.atributos["y"];
82    }
83  }
84
85  void EscenaJuego::manejar(EventoFlip& evento) {
86    objetosDelJuego.at(evento.atributos["idItem"])→flip(evento.atributos["flip"]);
87  }
88
89  void EscenaJuego::manejar(EventoCambioEstado& evento) {
90    objetosDelJuego.at(evento.atributos["idItem"])→asignarEstado(evento.atributos["
    estado"]);
91  }
92
93  void EscenaJuego::manejar(EventoEliminarItem& evento) {
94    objetosDelJuego.erase(evento.atributos["idItem"]);
95  }
96
97  void EscenaJuego::manejar(EventoRotacion& evento) {
98    objetosDelJuego.at(evento.atributos["idItem"])→asignarRotacion(evento.atributos
    ["angulo"]);
99  }
100
101 void EscenaJuego::manejar(EventoCreacionPersonaje& evento) {
102   this→miId = evento.atributos["idPersonaje"];
103   handler.setPlayerId(miId);
104 }
105
106 void EscenaJuego::recibirMiIdentificador() {
107   Evento* eventoCreacionPersonaje;
108   bool recibiId = false;
109   while(¬recibiId) {
110     recibiId = colaRecibir.get(eventoCreacionPersonaje);
111   }
112   eventoCreacionPersonaje→actualizar(*this);
113   delete eventoCreacionPersonaje;
114 }
115
116 void EscenaJuego::crearTerreno() {
117   int id, x, y, angulo;
118   YAML::Node escenaYaml = YAML::LoadFile("escenario.yaml");
119   YAML::Node objetos = escenaYaml["objetos"];
120
121   for (size_t i = 0; i < objetos.size(); ++i) {
122     id = objetos[i]["tipo"].as<int>();
123     x = conv.bloqueAPixel(objetos[i]["posX"].as<int>());
124     y = conv.bloqueAPixel(objetos[i]["posY"].as<int>());
125     angulo = objetos[i]["angulo"].as<int>();
126     VistaObjetoPtr vo = creadorTexturas.crear(id, x, y, angulo);
127
128     if (vo→getId() ≡ miId) {
129       //Centrar camara en mi jugador
```



```
130       int screenX, screenY;
131       window.getWindowSize(&screenX, &screenY);
132       deltaCamaraX = screenX/2 - x;
133       deltaCamaraY = screenY/2 - y;
134     }
135     objetosDelJuego.insert(std::make_pair(vo→getId(), vo));
136   }
137 }
138
139 void EscenaJuego::actualizarFondo() {
140   int xScreen, yScreen;
141   window.getWindowSize(&xScreen, &yScreen);
142   fondo.setDimensiones(xScreen, yScreen);
143   for(int i = 0; i < 5; ++i) {
144     for(int j = 0; j < 2; ++j) {
145       fondo.dibujarEn(-xScreen*2 + xScreen*i + deltaCamaraX/2,
146               -yScreen/2 + yScreen*j + deltaCamaraY/2);
147     }
148   }
149 }
150
151 EscenaJuego::~EscenaJuego() {}
```

jun 11, 19 16:12 **EscenaBase.h** Page 1/1

```
1  #ifndef ESCENA_BASE_H
2  #define ESCENA_BASE_H
3
4  #include "../Common/handler.h"
5
6  class EscenaBase : public Handler {
7  public:
8    virtual bool termino() = 0;
9    virtual void actualizar() = 0;
10   virtual void dibujar() = 0;
11   virtual int manejarEventos() = 0;
12   virtual ~EscenaBase() {}
13 };
14
15 #endif // ESCENA_BASE_H
```

jun 11, 19 16:12 **EnviadorEventos.h** Page 1/1

```
1  #ifndef ENVIADOR_EVENTOS
2  #define ENVIADOR_EVENTOS
3
4  #include "../Common/Thread.h"
5  #include "../Common/cola_bloqueante.h"
6  #include "../Common/Evento.h"
7  #include "../Common/Socket.h"
8
9  class EnviadorEventos : public Thread {
10 private:
11   ColaBloqueante<Evento*>& cola;
12   Socket& socket;
13   bool termino;
14 public:
15   EnviadorEventos(ColaBloqueante<Evento*>& cola, Socket& socket);
16   virtual void ejecutar();
17   void detener();
18 };
19
20 #endif
```

jun 11, 19 16:12 **EnviadorEventos.cpp** Page 1/1

```cpp
#include "EnviadorEventos.h"

EnviadorEventos::EnviadorEventos(ColaBloqueante<Evento*>& cola, Socket& socket)
:
  cola(cola), socket(socket) {
  termino = false;
}

void EnviadorEventos::ejecutar() {
  while(¬termino) {
    Evento* evento;
    cola.get(evento);
    evento→enviarPorSocket(socket);
    delete evento;
  }
}

void EnviadorEventos::detener() {
  socket.shutdown();
  termino = true;
  Thread::cerrar();
}
```

jun 11, 19 16:12 **CreadorTexturas.h** Page 1/1

```cpp
#ifndef CREADOR_TEXTURAS
#define CREADOR_TEXTURAS

#include "SdlWindow.h"
#include <memory>
#include "SdlTexture.h"
#include "VistaObjeto.h"

class CreadorTexturas {
private:
  int contadorID;
  SdlTexture bloqueTex;
  SdlTexture emisRecpTex;
  SdlTexture p1Tex, p2Tex, p3Tex, p4Tex;
  SdlTexture miscTex;
  SdlTexture puertaTex;
  SdlTexture efectosTex;
  SdlTexture pinToolTex;
  SdlTexture tortaTex;
  SdlTexture portalesTex;
public:
  CreadorTexturas(const SdlWindow& window);
  std::shared_ptr<VistaObjeto> crear(int tipo, int x, int y, int angulo);
private:
  void iniciarColores();
};

#endif
```

```cpp
#include "CreadorTexturas.h"
#include <iostream>
#include "../Common/Constantes.h"

#include "VistaBloqueMetal.h"
#include "VistaBloquePiedra.h"
#include "VistaAcido.h"
#include "VistaEmisor.h"
#include "VistaReceptor.h"
#include "VistaBoton.h"
#include "VistaPuerta.h"
#include "VistaPiedraMovil.h"
#include "VistaBarreraEnergia.h"
#include "VistaBolaEnergia.h"
#include "VistaPersonaje.h"
#include "VistaBloqueMetalDiagonal.h"
#include "VistaPinTool.h"
#include "VistaTorta.h"
#include "VistaPortalAzul.h"
#include "VistaPortalNaranja.h"


CreadorTexturas::CreadorTexturas(const SdlWindow& window) :
  bloqueTex("bloque_metal_diag.png", window),
  emisRecpTex("emisor_receptor.png", window),
  p1Tex("chell.png", window), p2Tex("chell.png", window),
  p3Tex("chell.png", window), p4Tex("chell.png", window),
  miscTex("miscelanea.png", window),
  puertaTex("puertas.png", window),
  efectosTex("efectos.png", window),
  pinToolTex("pin.png", window),
  tortaTex("cake.png", window),
  portalesTex("portales.png", window) {
    contadorID = 0;
    iniciarColores();
}

std::shared_ptr<VistaObjeto> CreadorTexturas::crear(int tipo, int x, int y, int angulo) {
  std::shared_ptr<VistaObjeto> vo;
  switch (tipo) {
    case ID_BLOQUE_METAL: vo.reset(new VistaBloqueMetal(bloqueTex)); break;
    case ID_BLOQUE_PIEDRA: vo.reset(new VistaBloquePiedra(bloqueTex)); break;
    case ID_ACIDO: vo.reset(new VistaAcido(miscTex)); break;
    case ID_EMISOR: vo.reset(new VistaEmisor(emisRecpTex, angulo)); break;
    case ID_RECEPTOR: vo.reset(new VistaReceptor(emisRecpTex, angulo)); break;
    case ID_BOTON: vo.reset(new VistaBoton(miscTex)); break;
    case ID_PUERTA: vo.reset(new VistaPuerta(puertaTex)); break;
    case ID_PIEDRA_MOVIL: vo.reset(new VistaPiedraMovil(efectosTex)); break;
    case ID_BARRERA_ENERGIA: vo.reset(new VistaBarreraEnergia(miscTex, angulo)); break;
    case ID_BOLA_ENERGIA: vo.reset(new VistaBolaEnergia(efectosTex, angulo)); break;
    case ID_PERSONAJE_1: vo.reset(new VistaPersonaje(p1Tex)); break;
    case ID_PERSONAJE_2: vo.reset(new VistaPersonaje(p2Tex)); break;
    case ID_PERSONAJE_3: vo.reset(new VistaPersonaje(p3Tex)); break;
    case ID_PERSONAJE_4: vo.reset(new VistaPersonaje(p4Tex)); break;
    case ID_BLOQUE_DIAGONAL_METAL: vo.reset(new VistaBloqueMetalDiagonal(bloqueTex, angulo)); break;
    case ID_PIN_TOOL: vo.reset(new VistaPinTool(pinToolTex)); break;
    case ID_TORTA: vo.reset(new VistaTorta(tortaTex)); break;
    case ID_PORTAL_AZUL: vo.reset(new VistaPortalAzul(portalesTex, angulo)); break;
    case ID_PORTAL_NARANJA: vo.reset(new VistaPortalNaranja(portalesTex, angulo)); break;
    default: throw std::runtime_error("Error obteniendo vista del objeto debido a id incorrecto");
```



```cpp
  }
  ++contadorID;
  vo→setPosInicial(x, y);
  vo→setId(contadorID);
  return vo;
}

void CreadorTexturas::iniciarColores() {
  p1Tex.setColor(255,64,64);
  p2Tex.setColor(96,96,255);
  p3Tex.setColor(64,255,64);
  p4Tex.setColor(255,255,64);
}
```

jun 11, 19 16:12 **Audio.h** Page 1/1

```
#ifndef AUDIO
#define AUDIO

#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>
#include <map>


class Audio {
private:
  Mix_Music* musicaFondo = NULL;
  Mix_Chunk* disparo = NULL;
  Mix_Chunk* salto = NULL;

  std::map<int, Mix_Chunk*> efectos;
public:
  Audio();
  ~Audio();
  void reproducirMusica();
  void pararMusica();
  void reproducirEfecto(int idEfecto);
};

#endif
```

jun 11, 19 16:12 **Audio.cpp** Page 1/1

```
#include "Audio.h"
#include "../Common/Constantes.h"

Audio::Audio() {
  if(SDL_Init(SDL_INIT_AUDIO) < 0) {
    printf("SDL could not initialize! SDL Error: %s\n", SDL_GetError());
  }
  if(Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048) < 0) {
    printf("SDL_mixer could not initialize! SDL_mixer Error: %s\n", Mix_GetError());
  }

  musicaFondo = Mix_LoadMUS("musica_fondo.mp3");
  if(musicaFondo == NULL) printf("%s\n", Mix_GetError());

  disparo = Mix_LoadWAV("disparo_portal.wav");
  if (disparo == NULL) printf("%s\n", Mix_GetError());
  efectos.insert(std::make_pair(EFECTO_DISPARO, disparo));

  salto = Mix_LoadWAV("salto.wav");
  if (salto == NULL) printf("%s\n", Mix_GetError());
  efectos.insert(std::make_pair(EFECTO_SALTO, salto));

}

void Audio::reproducirMusica() {
  Mix_VolumeMusic(MIX_MAX_VOLUME/10);
  if(Mix_PlayingMusic() == 0) {
    Mix_FadeInMusic(musicaFondo, -1, 3000);
  }
}

void Audio::pararMusica() {
  Mix_FadeOutMusic(1000);
}

void Audio::reproducirEfecto(int idEfecto) {
  Mix_Chunk* efecto = efectos.at(idEfecto);
  Mix_PlayChannel(-1, efecto, 0);
}

Audio::~Audio() {
  Mix_FreeMusic(musicaFondo);
  musicaFondo = NULL;
  for (auto& it : efectos) {
    delete it.second;
  }
  Mix_Quit();
}
```

jun 11, 19 16:12 **Area.h** Page 1/1

```
1  #ifndef __AREA_H_
2  #define __AREA_H_
3
4  class Area {
5  public:
6    Area();
7    void set(int x, int y, int width, int height);
8    Area(int x, int y, int width, int height);
9    int getX() const;
10   int getY() const;
11   int getWidth() const;
12   int getHeight() const;
13   bool estaAdentro(int x, int y);
14 private:
15   int x, y;
16   int width, height;
17 };
18
19 #endif
```

jun 11, 19 16:12 **Area.cpp** Page 1/1

```
1  #include "Area.h"
2  #include <iostream>
3
4  Area::Area() {}
5
6  void Area::set(int x, int y, int width, int height) {
7    this→x = x; this→y = y;
8    this→width = width; this→height = height;
9  }
10
11 Area::Area(int x, int y, int width, int height) :
12   x(x), y(y), width(width), height(height){
13 }
14
15 int Area::getX() const {
16   return this→x;
17 }
18 int Area::getY() const {
19   return this→y;
20 }
21 int Area::getWidth() const {
22   return this→width;
23 }
24 int Area::getHeight() const {
25   return this→height;
26 }
27
28 bool Area::estaAdentro(int x, int y) {
29   if (this→x > x ∨ x > (this→x + this→width)) return false;
30     if (this→y > y ∨ y > (this→y + this→height)) return false;
31   return true;
32 }
```

jun 11, 19 16:12 **AnimacionPersonaje.h** Page 1/1

```
#ifndef ANIMACION_PERSONAJE
#define ANIMACION_PERSONAJE

#include "Area.h"
#include "SdlTexture.h"
#include <vector>
#include <map>
#include <string>

class AnimacionPersonaje {
private:
  std::map<int, std::vector<Area>> mapaAnimaciones;
  int actualSize;
public:
  AnimacionPersonaje();
  Area obtenerConEstado(int estado, int frame);
  int size() const;
};

#endif
```

jun 11, 19 16:12 **AnimacionPersonaje.cpp** Page 1/1

```
#include "AnimacionPersonaje.h"
#include "../Common/Constantes.h"

AnimacionPersonaje::AnimacionPersonaje() {
  //Idle
  std::vector<Area> clipsIdle;
  for (int i = 0; i < 7; ++i) {
    Area area(i*105, 2074, 105, 215);
    clipsIdle.push_back(area);
  }
  mapaAnimaciones.insert(std::make_pair(ESTADO_IDLE, clipsIdle));

  //Corriendo
  std::vector<Area> clipsCorriendo;
  for (int i = 0; i < 8; ++i) {
    Area area(1 + i*195, 4123, 195, 215);
    clipsCorriendo.push_back(area);
  }
  for (int i = 0; i < 4; ++i) {
    Area area(1 + i*195, 4330, 195, 215);
    clipsCorriendo.push_back(area);
  }
  mapaAnimaciones.insert(std::make_pair(ESTADO_CORRIENDO, clipsCorriendo));

  //Disparando
  std::vector<Area> clipsDisparando;
  for (int i = 0; i < 5; ++i) {
    Area area(i*173, 2305, 173, 215);
    clipsDisparando.push_back(area);
  }
  mapaAnimaciones.insert(std::make_pair(ESTADO_DISPARANDO, clipsDisparando));

  //Muerto
  std::vector<Area> clipsMuerto;
  for (int i = 0; i < 8; ++i) {
    for (int j = 0; j < 9; ++j) {
      Area area(1656 + 176*j, 3407 + 265*i, 170, 215);
      clipsMuerto.push_back(area);
    }
  }
  mapaAnimaciones.insert(std::make_pair(ESTADO_MUERTO, clipsMuerto));

  //Saltando
  std::vector<Area> clipsSaltando;
  for (int i = 0; i < 4; ++i) {
    Area area(i*158, 1836, 155, 215);
    clipsSaltando.push_back(area);
  }
  mapaAnimaciones.insert(std::make_pair(ESTADO_SALTANDO, clipsSaltando));
}

Area AnimacionPersonaje::obtenerConEstado(int estado, int frame) {
  std::vector<Area> ret = mapaAnimaciones.at(estado);
  actualSize = ret.size();
  return ret.at(frame);
}

int AnimacionPersonaje::size() const {
  return actualSize;
}
```

jun 11, 19 16:12 **Table of Content** Page 1/3

Table of Contents

jun 11, 19 16:12 **Table of Content** Page 2/3

jun 11, 19 16:12 **Table of Content** Page 3/3