FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Argentum - Trabajo Final
Manual Técnico

Taller de Programación 1
Primer cuatrimestre de 2020

Alumno	Número de Padrón	Email
Ruiz, Francisco	99429	fran7ruiz9@gmail.com
Belosevich, Victor	97757	victor.belosevich94@gmail.com

Índice

1. Requerimientos de software

El juego fue desarrollado en Ubuntu 19.10 o mayor y puede correrse en distribuciones Debian. Para desarrollar y ejecutar el juego es necesario tener instalado las siguientes librerías:

- SDL2 versión 2.0.10
- SDL2-image versión 2.0.5
- SDL2-mixer versión 2.0.4
- SDL2-ttf versión 2.0.15

Para compilar e instalar el mismo, se utilizó la herramienta *Cmake* versión 3.14 o mayor. La otra librería externa utilizada es *RapidJson*, para la carga del mapa y el archivo de configuración con la versión 1.1.0, que ya se encuentra incluida en el proyecto. Por último, el compilador g++ utilizado es versión 9.2.1 y gdb, utilizado para debugging, en su versión 8.3.

2. Descripción general

Argentum se dividió en 2 módulos, el servidor y el cliente.

El servidor es el encargado de levantar el mapa del mundo de Argentum, la configuración del juego, ambos desde archivos json, y mantener el estado del mismo todo el tiempo que este se esté ejecutando. Es capaz de manejar múltiples clientes y a medida que cada uno de ellos se va del juego, elimina a su personaje del mapa.

El cliente es una aplicación gráfica que le permite al usuario desenvolverse en las tierras de Argentum. Al iniciar, por líneas de comando indicará la raza, clase para su jugador, el host y el puerto al cual debe conectarse. Una vez establecida la conexión, el servidor enviará la información del mapa estático del juego y el primer "paquete" con la información propia del personaje del jugador.

Una vez que ya el jugador ha sido inicializado en el mundo, el servidor le irá notificando al cliente cada cierto intervalo de tiempo, el estado actual del mundo e incluso el estado actual del personaje del jugador. El cliente procesará estos mensajes recibidos y renderizará por pantalla todas las novedades que le fueron informadas.

El juego seguirá funcionando y los personajes de cada uno de los clientes estarán presentes en el mapa, siempre y cuando estos sigan jugando. Una vez que el cliente se desconecta y cierra su ventana, el servidor sacará del mapa a su personaje.

3. Cliente

3.1. Descripción general

Como mencionamos previamente, el cliente es una aplicación de interfaz gráfica. Esta interfaz fue desarrollada en SDL2, con el uso de sus librerías adicionales como SDL2-image, SDL2-mixer y SDL2-ttf.

El cliente recibe 4 parámetros de entrada desde la línea de comandos. Estos parámetros, nos permitirán conocer la raza, clase, host y puerto respectivamente.

Una vez, que se haya establecido la comunicación, el juego nos dará la bienvenida con una pantalla de "carga" que al presionar la tecla *ENTER* nos permitirá comenzar a jugar.

3.1.1. Estructura

El cliente se divide en 3 threads:

- Gameloop (renderer)
- Receiver
- Dispatcher

El hilo principal del cliente es el Gameloop que será el encargado de establecer la conexión inicial con el servidor. Esta conexión está basada en 3 pasos que se explicarán en la sección *Protocolo de Comunicación*.

Esta conexión se hace a través de un socket que estará encapsulado dentro de una clase llamada *Communication-Protocol*. Esta clase nos permite que varios hilos puedan trabajar con el mismo socket y que no quede en un estado inválido, ya que la implementación de nuestro *Socket* al pasarlo como referencia, se pierde el *ownership* del mismo.

Una vez que haya obtenido la información de su personaje y el mapa estático del mundo de Argentum, el gameloop lanzará a correr a los otros 2 threads, pasandole la instancia de *CommunicationProtocol* como referencia dado que estos serán los encargados de recibir el estado del mundo y de enviar los inputs del jugador al servidor, respectivamente.

A medida que vaya pasando el juego, el gameloop irá retirando de una queue de mensajes,*DataQueue*, la información del servidor para actualizar su modelo en consecuencia, y a partir de ahí renderizarlo según corresponda. Esta *DataQueue* no es bloqueante al realizar el pop y de esta manera se evita que el gameloop quede bloqueado esperando información del servidor y no pueda seguir renderizando o atendiendo inputs del jugador. Toda esta información es recibida por el hilo Receiver que los pusheará en la queue recién mencionada.

Por otro lado, el gameloop irá atendiendo a los inputs que realice el usuario, estos pueden ser por teclado o por eventos de mouse, dependiendo de la actividad que realice. Estos inputs serán enviados, o no, al servidor, que es el encargado de resolver la parte lógica de los inputs, dependiendo del estado del personaje del cliente. Todos los inputs son pusheados en una queue de comandos, *InputQueue*, que el hilo Dispatcher irá vaciando bloqueándose en el pop en caso de que esté sin ningún input. Si el input que obtuvo al realizar el pop es distinto a nada, procederá a encodear el comando para enviarlo al servidor. En caso contrario, ignorará el input del jugador.

Por último, una vez actualizado el estado del mundo, renderizarlo y haber manejado todos los eventos del jugador, el hilo principal "dormirá" el tiempo necesario para completar el loop en un tiempo estipulado como **gameloop time**.

3.1.2. Estados del personaje

Para manejar esta parte del diseño se utilizó el patrón **State**. Esto nos permitió enfrentarnos a los diferentes comandos disponibles en el juego y que cada uno actué de manera polifórmica dependiendo del estado del personaje. De esta manera, evitamos saturar al servidor con información inútil y que haría más lento la ejecución de las órdenes. Un ejemplo de esto, es el estado *ResurrectState* en los cuales ningún tipo de input es válido ya que el personaje, hecho un fantasma se desplaza automáticamente por el mapa hasta el sacerdote más cercano para que lo reviva. Pero a diferencia del estado anterior, el estado *StillState* permitirá ingresar casi cualquier tipo de acción, salvo aquellas que requieren de estar interactuando con algún profesional de las ciudades.

Hay ciertos estados que se verán en el servidor, como por ejemplo *EquipStateCharacter*, que son absorbidos por el estado *StillState* en el cliente.

El estado del jugador es enviado desde el servidor cada cierto intervalo de tiempo, dependiendo en cuál esté puede tener alguna animación o sonido característico, por ejemplo si el jugador se encuentra en el estado *MeditateState* se verá el efecto de esta acción.

3.1.3. Wrappers de SDL

Para mejorar las prestaciones de SDL, se realizaron wrappers RAII para las ventanas, texturas, fuentes y sonidos. Esto nos permitió trabajar con mayor comodidad y no depender de hacer llamadas a funciones en cualquier lado del código para liberar memoria utilizada por las clases y estructuras de SDL.

Además, dado que la carga de una imagen o sonido es costosa, se decidió generar 2 clases encargadas de realizar la carga de las mismas al inicio del cliente, el *MusicManager* y el *TextureManager*. Éstas clases sirven como elementos de consulta para las demás que requieran de una determinada textura para renderizarse o un sonido para acompañar la animación.

3.1.4. Renderización

La renderización del juego está principalmente distribuida en 2 clases, la *UI* y la *Camera*. La primera, es la encargada de mostrar por pantalla, el estado general del jugador, ya sea su vida, su inventario, su experiencia o su equipamiento. A su vez, en caso de que el jugador esté interactuando con algún NPC, también renderizará las acciones e items que estos nos dispongan.

Por otro lado, la clase *Camera* es la vista de águila centrada en el jugador objetivo para este cliente. Todas las texturas se renderizarán según su posición relativa a la posición de la cámara. Ambas tienen al jugador como atributo, dado que es necesario para poder mostrar la información actual al usuario.

3.2. Clases

A continuación se listarán las principales clases utilizadas dentro del cliente:

- **Game:** Es el hilo principal del cliente. Al crearse, inicializa la ventana donde se va a renderizar el juego, el hilo Dispatcher y el hilo Receiver, las queues que utilizará para comunicarse con estos 2, y el *CommunicationProtocol* que como ya vimos tiene al socket dentro de su clase. Tiene un método *init* que recibe los parámetros del juego y los utiliza para realizar la conexión inicial con el servidor, cargar los assets, generar la UI y la Camera. Por último, el método *run* es propiamente el gameloop del juego, del lado del cliente.

- **Dispatcher:** Hereda de la clase *Thread*. Al crearse recibe el *CommunicationProtocol*, que contendrá el socket ya conectado al servidor y la *InputQueue* que utilizará para comunicarse con el Game. Además, encodeará a los *InputInfo*, que obtiene de la queue, para enviarlos al servidor, siempre y cuando el input sea un comando válido.
- **Receiver:** Hereda de la clase *Thread*. Al crearse recibe el *CommunicationProtocol*, que contendrá el socket ya conectado al servidor y la *DataQueue* que utilizará para comunicarse con el Game, pusheando los *Message* recibidos desde el servidor.
- **CharacterState:** Esta clase es utilizada por los personajes del juego para retratar su estado. Las implementaciones de esta clase abstracta permitirán o no el efecto de ciertos comando. Cada uno de los estados se identifica con un *CharacterStateID*. Las clases que heredan de CharacterState son:
  - **AttackState**
  - **InteractState**
  - **MeditateState**
  - **MoveState**
  - **ResurrectState**
  - **StillState**
- **Character:** Es una clase abstracta que representa a los personajes presentes en el mundo de Argentum. Posee una referencia al *TextureManager* y al *MusicManager*, un id, y su posición. Además tiene como atributo una *Direction*, un *Body*, *Head*, *Helmet*, *Shield*, *Weapon* y *Animation*. La renderización de los mismos se hacen en una posición relativa a la cámara del juego centrada en el *Player*.
- **Player:** Es el jugador del cliente. Hereda de *Character*. Tiene como atributo a un *PlayerInfo* que es donde viene la información desde el servidor. Tiene un método *update* que actualiza las animaciones de los frames equipados y otro método llamado *updatePlayerInfo* que recibe un PlayerInfo y que actualiza el estado general del Player para sincronizarlo con el servidor. Por último, es capaz de manejar los inputs del usuario según el *CharacterState* en el cuál se encuentre.
- **NPC:** Hereda de *Character*. Representa a todos los demás personajes o items dropeados en el mapa que haya en Argentum. Tiene una funcionalidad similar a Player, sin poder manejar los inputs de usuario. Si se trata de una criatura, dependiendo de su *Body* también se asignará algún sonido característico.
- **Item:** Es una clase muy simple que solo conoce su textura, el ancho y el alto de la misma. Sabe renderizarse en la posición indicada. Es utilizada para renderizar en el mapa a aquellos objetos resultantes de algún drop.
- **Body:** Hereda de *Item*. Es una clase abstracta de la cual heredan todos los cuerpos o armaduras disponibles en Argentum. Contiene una *Direction* y conoce la cantidad de frames de la animación que se le serán seteados por las respectivas implementaciones. Además, es la encargada de realizar el *update* de las animaciones de este estilo. Todos los Body son identificados con id del tipo *BodyID*:
  - **BlueCommonBody**
  - **RedCommonBody**
  - **GreenCommonBody**
  - **BlueTunic**
  - **LeatherArmor**
  - **PlateArmor**
  - **BankerBody**
  - **PriestBody**
  - **MerchantBody**
  - **GhostBody**
  - **GoblinBody**
  - **SpiderBody**
  - **SkeletonBody**
  - **ZombieBody**

- **Head:** Hereda de *Item*. Es el equivalente de Body para las cabezas que representan a las distintas razas disponibles en el juego. Se identifican con un id del tipo *HeadID*:
  - **ElfHead**
  - **GnomeHead**
  - **DwarfHead**
  - **HumanHead**
  - **PriestHead**
  - **ZombieHead**
- **Helmet:** Hereda de *Item*. Es el equivalente de Head para los cascos que están disponibles en el juego. Se identifican con un id del tipo *HelmetID*:
  - **MagicHat**
  - **Hood**
  - **IronHelmet**
- **Shield:** Hereda de *Item*. Es el equivalente de Body para los escudos que están disponibles en el juego. Se identifican con un id del tipo *ShieldID*:
  - **TurtleShield**
  - **IronShield**
- **Weapon:** Hereda de *Item*. Es el equivalente de Body para las armas que están disponibles en el juego. Se identifican con un id del tipo *WeaponID*:
  - **AshStick**
  - **Ax**
  - **CompoundArc**
  - **Crosier**
  - **SimpleArc**
  - **Hammer**
  - **LongSword**
  - **GnarledStick**
  - **ElficFlaute**
- **Animation:** Muy similar a las anteriores pero representa a los efectos visuales y auditivos que se producen en el juego, estos pueden ser producidos por ataques de armas o por acciones que genere el jugador. Tienen una textura y un efecto que serán ejecutados en loop o no, dependiendo de su implementación en las clases que hereden de esta. Algunas de las implementaciones son:
  - **CureAnimation**
  - **ExplotionAnimation**
  - **HitAnimation**
  - **MagicArrowAnimation**
  - **MeditateAnimation**
  - **MissileAnimation**
- **UI:** Clase encargada de renderizar el estado general de todo el jugador. Tiene un puntero al jugador del cual debe mostrar las estadísticas, ya sea por su inventario o cantidad de vida, oro, etc o el equipamiento actual del jugador. Además, en caso de que el jugador esté en estado *Interact* se mostrará por pantalla la interfaz correspondiente al NPC con el cuál se esté interactuando. Esto se realiza delegándolo en clases que heredan de *NPCInterface* como lo pueden ser *PriestInterface*, *BankerInterface* o *MerchantInterface* donde cada una de ellas mostrará con el layout correspondiente la información que este profesional tenga para nosotros. Esta información es obtenida a través de un objeto del tipo *NPCInfo* que es enviado por el servidor en caso de estar interactuando.

- **Button:** Son los distintos tipos de botones que se pueden apreciar en la interfaz. Esta es una clase abstracta que tiene diferentes implementaciones como un *SelectButton* utilizado para seleccionar items del inventario o *RaisedButton* utilizado para generar acciones dependiendo de qué botón se trate, que llamarán al handle del *Player* correspondiente a ese evento.
- **Camera:** Es la clase destinada a mantener la cámara del juego centrada sobre el jugador que es seteado como objetivo. Al momento de crearla, tiene conocimiento de las dimensiones del mapa para no traspasar los límites del mismo. Si se le da un punto dado puede informar la distancia que tiene al objetivo de la cámara. Esto fue hecho para poder manejar los sonidos de ciertos NPC y que estos estén relativamente cerca para ejecutar su efecto sonoro.
- **Presentation:** Esta clase es la pantalla de carga del juego. Esperará que se ingrese un *ENTER* para continuar.
- **GameMap:** Es el mapa estático del juego. Está compuesto por un vector de *Tile* y un map que tiene todos las texturas utilizadas para renderizar el mapa. Es capaz de dibujar las distintas capaz que lo componen, diferenciando el suelo de aquellos objetos que están en capaz superiores, como casas y árboles.
- **Window:** Es el wrapper correspondiente a una ventana de SDL. Encapsula la creación, manipulación y liberación de los recursos asociados. Puede realizar el manejo de eventos correspondientes al tamaño de la ventana.
- **Tile:** Contiene una *Texture* y las dimensiones de la misma. Además conoce la posición en la cual debe ser dibujada. Es utilizada para renderizar el mapa estático de Argentum.
- **Texture:** Es el wrapper correspondiente a una textura de SDL. Encapsula la creación, manipulación y liberación de los recursos asociados. Tiene una referencia al renderer que la puede utilizar. Además posee un método *render* que recibe 2 SDLRect que permitirán renderizar una porción de textura en una porción del Renderer.
- **TextureManager:** Es la clase que contiene a todas las *Texture* que se cargan al iniciar el juego. Las identifica con un *TextureID* y puede ser consultada desde varios métodos *getTexture* que devuelve como referencia la textura correspondiente al parámetro indicado. En caso de que este parámetro no sea un TextureID, y sea, por ejemplo un *BodyID*, realizará una conversión para encontrar la textura correspondiente.
- **Music:** Es el wrapper correspondiente a un MixMusic de SDL. Encapsula la creación, manipulación y liberación de los recursos asociados. Es utilizado para reproducir la música de fondo del juego.
- **Effect:** Es el wrapper correspondiente a un MixChunk de SDL. Encapsula la creación, manipulación y liberación de los recursos asociados. Es utilizado para la carga y reproducción de los efectos de sonido de las animaciones o sonido ambiente que contiene el juego.
- **MusicManager:** Análogo al *TextureManager* pero en vez de Texture contiene objetos de tipo *Music* y *Effect*. A todos los identifica por un *MusicID* que si es pasado como parámetro a los métodos *getMusic* y *getEffect* se obtendrá una referencia constante a dicho elemento.
- **Font:** Es el wrapper correspondiente a una TTFFont. Encapsula la creación, manipulación y liberación de los recursos asociados. Al construirla, se genera a partir de un path, con un tamaño y color determinado que luego podrá modificarse. Permite crear texturas con un determinado texto. Estas texturas quedarán en responsabilidad del usuario en ser liberadas.

A continuación se listarán las clases comunes para ambas aplicaciones:

- **GameObjectInfo:** Es una clase destinada a informar el estado actual de cualquier personaje u objeto no estático del mundo de Argentum. Es la información utilizada por el NPC del cliente para renderizarse como corresponda. Contiene un id, la posición, la dirección a la cual está orientado, un atributo que representa las texturas que componen al objeto, como por ejemplo que BodyID y HeadID tiene. También indica el *CharacterStateID*, y si está siendo atacado, se indica el WeaponID, esto es para renderizar la animación correspondiente.
- **PlayerInfo:** Hereda de *GameObjectInfo*. Muy similar en su propósito pero es exclusivo para el personaje del cliente al cual el servidor se lo envía. Se informa la vida, el mana, el inventario, entre otras cosas, información solamente útil para el personaje del cliente.
- **InputInfo:** Es un struct utilizado para encapsular los inputs del usuario. Tiene como atributos el InputID que sirve para identificar la orden, un aditional que puede contener cualquier información extra dependiendo el comando y un Point, por si se trata de un input que requiera una posición en el mapa.

- **NPCInfo:** Es un struct utilizado para encapsular la información que tiene para brindarnos el NPC con el cual estamos interactuando. Contiene el tipo de NPC con el que interactuamos, las acciones que puede realizar, los items que tiene, etc.
- **Message:** Es un wrapper del mensaje recibido por le socket y es generado por el *CommunicationProtocol*. Conformado por un vector de uint8 que almacena la información recibida, la longitud del mismo y el tipo de mensaje que es. Cada tipo de mensaje se explicará más en detalle en la sección **Protocolo de Comunicación**. Además brinda un mecanismo de lectura para ir avanzando por los bytes.
- **BlockingQueue:** Es un template de una cola bloqueante que permite un acceso controlado a los recursos encolados. En su constructor tiene un booleano para indicar si es necesario que se bloquee al realizar el pop.
- **DataQueue:** Especificación de la BlockingQueue para datos de tipo *Message*.
- **InputQueue:** Especificación de la BlockingQueue para datos de tipo *InputInfo*.
- **Socket:** Clase que encapsula el comportamiento basico de un socket. Realiza el bind, accept, connect, send, recieve, etc. Al pasarlo por referencia se pierde el ownership del mismo. No tiene habilitados los constructores por copia.
- **CommunicationProtocol:** Es un wrapper del Socket mencionado anteriormente. Surgió por la necesidad de tener una clase que permita utilizar el mismo socket por diferentes hilos de ejecución, para poder mantener una mejor comunicación sin que se que el hilo principal quede bloqueado. Tiene como métodos *connect*, *send* que wrappean los métodos del *Socket* y un méotdo *recieve* que retorna un *Message* con la longitud, el tipo de mensaje y el mensaje en sí.
- **Decoder:** Clase estática capaz de encodear y decodear las diferentes cosas que se envían en el protocolo de comunicación diseñado para Argentum. En la sección **Protocolo de Comunicación** ahondaremos más en esta clase.
- **Chrono:** Clase diseñada para devolver la cantidad de milisegundos que hubo entre dos llamadas a la función *lap*.
- **Point:** Clase diseñada para manejar coordenadas (x,y). Sobrecarga diversos operadores que permiten manejar de mejor manera las posiciones de los jugadores dentro del mundo de Argentum.
- **Random:** Clase estática diseñada para generar números aleatorios entre intervalos, estos pueden ser float o int.
- **Thread:** Clase abstracta y que encapsula el comportamiento de un thread. Tiene implementado los métodos *start* y *join*, dejando a sus clases hijas la implementación del método *run*.
- **ObjectLayer:** Clase que se utiliza para almacenar las posiciones estáticas de los objetos dentro del mapa.
- **JsonReader:** Clase estática que recibe un archivo .json y lo serializa a un objeto json que luego será parseado.
- **StaticObject:** Tiene las dimensiones de los objetos estáticos como las casa, los NPC, los árboles y los puntos de nido desde donde aparecen las criaturas. Esta clase es utilizada en la creación del Board para localizar las celdas que están ocupadas con objetos estáticos.
- **TiledMap:** Clase utilizada como DTO que tiene la información estática del mapa creado con la herramienta Tiled. Esta clase se encodea en el servidor y es desencodeada en el cliente para su posterior transformación a la clase GameMap. El proceso que se realiza para encodear y desencodear el mapa se explica en detalle en la sección Protocolo de Comunicación.
- **TileLayer:** Clase que contiene información sobre la distribución de los tiles en el mapa. Posee los id de cada tile que luego van a ser utilizados en conjunto con la clase TileSet para el renderizado del mapa.
- **TileSet:** Clase que contiene el path a la imagen que en el cliente se utilizará para generar la textura correspondiente. También tiene el firstgid que cumple el propósito de vincular cada tileset con el tile correspondiente.

3.3. Diagramas UML

El siguiente diagrama nos permite observar como está compuesta la clase *Game* y como se relacionan los threads *Dispatcher* y *Receiver* con ella. A través de las respectivas colas bloqueantes y utilizando el mismo *Communication-Protocol* compuesto por un *Socket*.

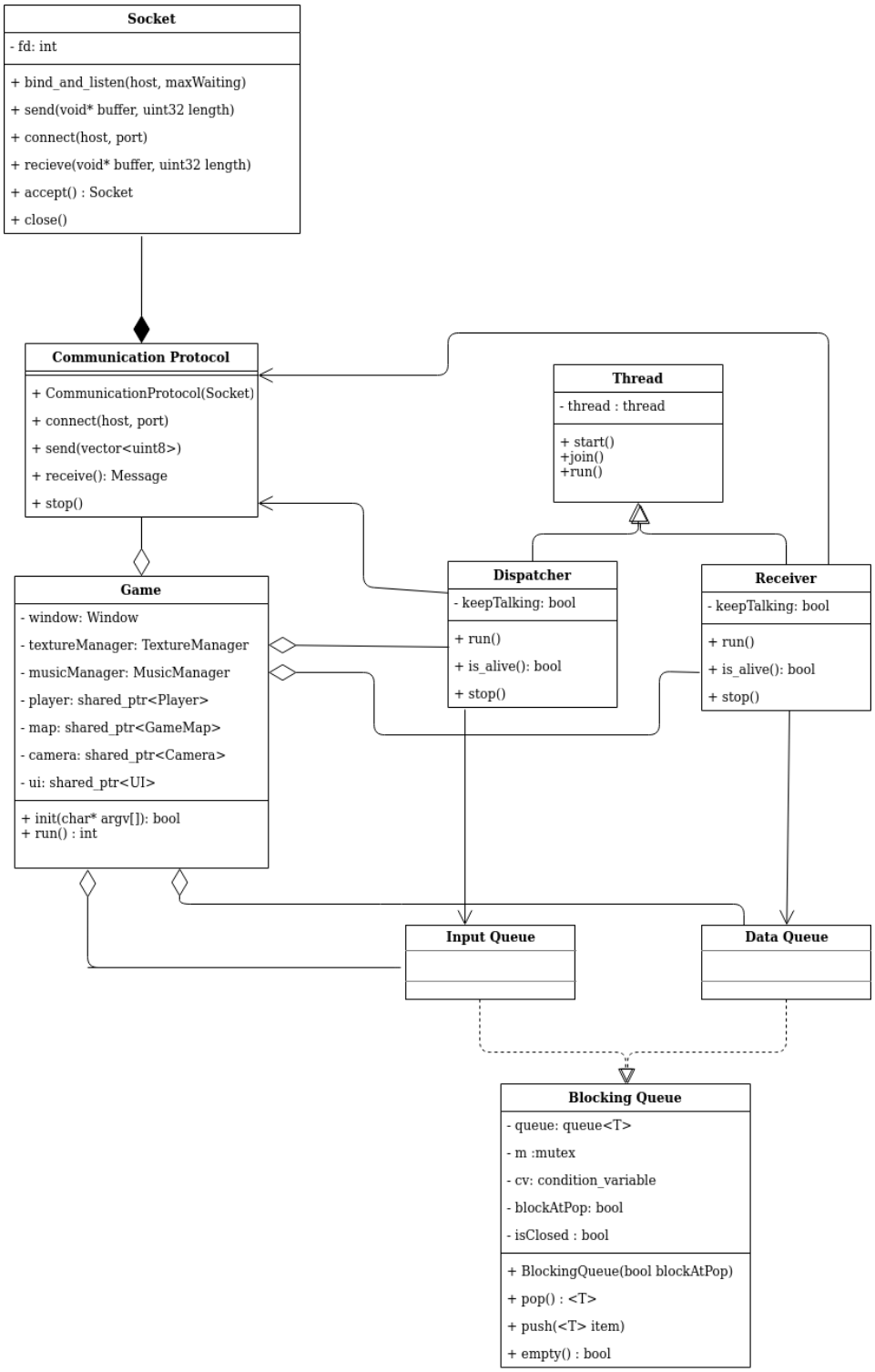

Figura 1: Diagrama Clase Game

A continuación, veremos un diagrama de secuencia que nos permite entender la conexión inicial desde el punto de vista del cliente.

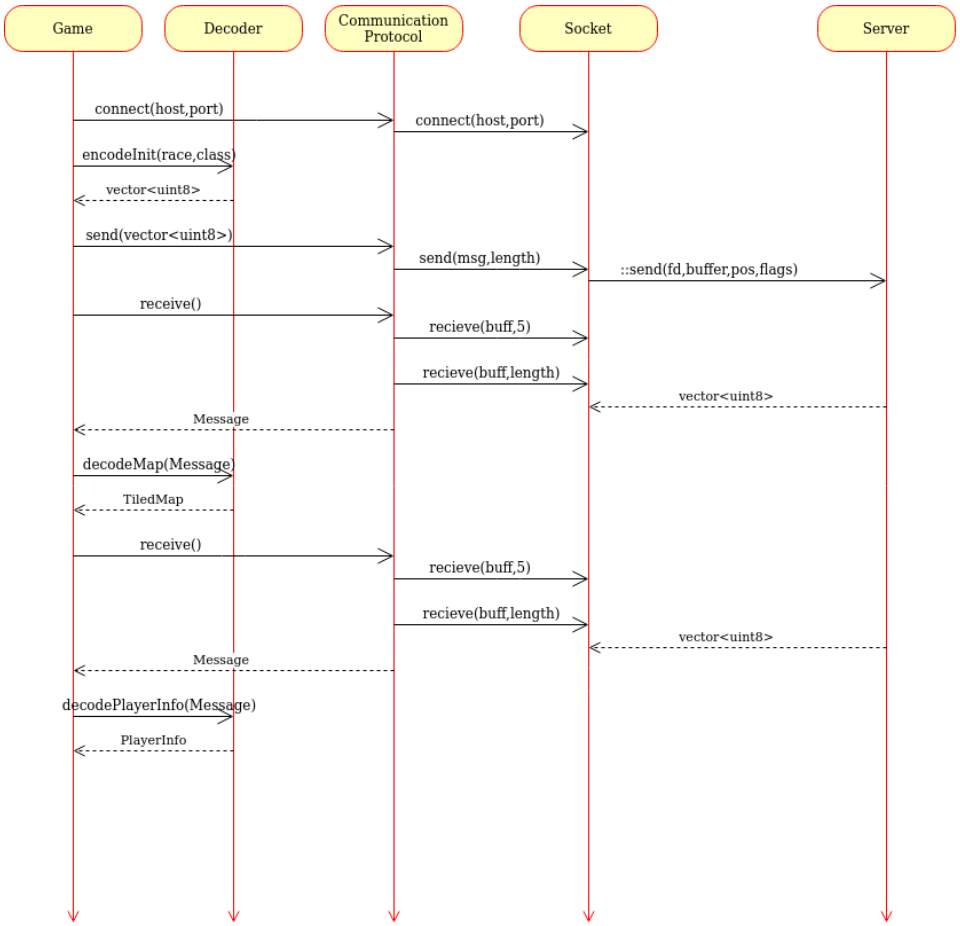

Figura 2: Diagrama Conexión Inicial

Una vez sucedidos los pasos explicados en la Figura 2, es cuando el hilo principal, lanza a correr los hilos Dispatcher y Receiver pasándoles como referencia al *CommunicationProtocol* que contiene el *Socket* conectado al servidor.

4. Servidor

4.1. Descripción general

El servidor recibe un único parámetro, el archivo de configuración, que es donde se encuentra entre otras cosas, las descripciones de las distintas razas, criaturas y clases. También nos brinda información sobre el puerto en donde el servidor estará aceptando nuevos clientes.

Una vez que el servidor haya cargado el archivo de configuración y el mapa, ambos archivos *.json*, procede a lanzar los thread World y PlayerAcceptador y se queda esperando un carácter por entrada estándar. En caso que dicho carácter sea la letra 'q', el hilo procede a cerrar de forma ordenada los otros 2 hilos y se cierra.

4.1.1. Estructura

El servidor se divide en 3 threads:

- Main
- World
- PlayerAcceptor

El hilo World, se encarga de inicializar el mapa creado previamente con Tiled y de tenerlo en memoria, para poder pasarlo a los clientes que se conectan. El hilo PlayerAcceptor, como bien indica su nombre, es el encargado de aceptar nuevos clientes que son insertados en el World, el cual notifica los eventos que ocurren y recibe las acciones de todos los clientes.

4.1.2. Proceso para aceptar un nuevo cliente

Cada vez que se detecta una nueva conexión, el acceptor lanza un ThLobbyPlayer, al cual se le pasa una referencia al World y por lo tanto cuando recibe un mensaje de inicio del cliente es el encargado de que se cree un nuevo personaje con la raza y la clase especificados en el mensaje. También se encarga de enviar tanto el mapa como la información del personaje creado. El World añade este nuevo cliente a su lista de jugadores activos y mantendrá comunicación con el mismo, hasta que se cierre el servidor o el cliente abandone la partida. Luego de realizados estos pasos, se realiza un join al ThLobbyPlayer y queda a la espera de ser eliminado por el PlayerAcceptor. La comunicación con el World se da a través del ThPlayer.

4.1.3. Inicio del World y carga del Board

El World al iniciarse, carga el archivo *finishedMap.json* y con ayuda del JsonReader genera un objeto json que es utilizado por la clase TiledMap para crear el mapa. Luego se instancian las distintas profesiones y con el TiledMap anteriormente cargado y el archivo de configuración, se procede a inicializar el Board. El Board hace uso de las dimensiones del mapa para determinar la cantidad de celdas a crear y con la información obtenida en los objectLayers, determina que celdas serán ocupadas por los objetos estáticos del mapa (Construcciones, árboles y murallas) con los que los personajes puedan colisionar. También configura las celdas que pertenecen a una de las ciudades o a nidos de criaturas. Por último el World se encarga de colocar a los npc en sus respectivas posiciones iniciales y de agregar a las criaturas en los distintos nidos hasta que se llegue al límite establecido en el archivo de configuración.

4.2. Clases

A continuación se listarán las principales clases utilizadas dentro del servidor:

- **World:** Una vez inicializada la clase, por el proceso explicado anteriormente, se encarga de realizar el update periódico de todos los gameObjects e informar a los distintos clientes los eventos ocurridos durante ese periodo de tiempo. También gestiona la creación y destrucción de los de las criaturas y de los ítems generados a partir de la destrucción de las mismas.
- **PlayerAcceptor:** Hereda de *Thread* y el el hilo responsable de ir aceptando a los nuevos jugadores que se conecten al servidor. A medida que acepta a un nuevo cliente, lanzará a correr a otro thread llamado *ThLobbyPlayer*.
- **ThLobbyPlayer:** Hereda de *Thread*. Es el encargado de realizar la comunicación inicial con el cliente e indicarle al mundo la raza y clase que este eligió. A su vez enviará el mapa estático y su primer PlayerInfo.
- **ThPlayer:** Clase encargada de enviar la información actualizada del mundo al cliente. Esto no se realiza todo el tiempo, si no que se hace periódicamente cuando el servidor realiza el update de los gameObjects. La información que se envía al cliente está compuesta por un PlayerInfo, una lista de todos los gameobjects del juego, sin contar al jugador correspondiente al cliente en cuestión, y se envía un NPCInfo en caso de que el jugador esté interactuando con un NPC. La comunicación se efectúa utilizando la clase Communication Protocol y el Decoder. Los distintos mensajes enviados, serán detallados en la sección Protocolo de Comunicación.
- **ThPlayerReceiver:** Es la clase encargada de recibir los inputs del cliente y los encola en la InputsQueue del gameCharacter asociado al ThClient. De esta forma el personaje cuando termina un estado, utiliza estos inputs para modificar su estado.
- **Board:** Clase encargada de contener a las celdas del juego y realizar operaciones entre ellas. Algunas de las operaciones que nos permite realizar son:
  - Obtener la distancia entre 2 celdas, para eso se utilizó la distancia Manhattan.
  - Obtener una celda a partir de un punto y viceversa.
  - Obtener las celdas adyacentes a una celda.
  - Obtener la siguiente celda, dada una dirección de movimiento.
  - Obtener el camino mínimo entre 2 celdas, utilizando el algoritmo A*.
- **NestContainer:** Clase que contiene información de los distintos nidos que se encuentran por el mapa. Permite obtener el nido con menos cantidad de criaturas de forma eficiente.
- **Nest:** Es la clase encargada de configurar los límites por donde se pueden mover las criaturas que fueron creadas en ese nido. Almacena los ids de las criaturas que posee y resulta útil a la hora de notificar la presencia de un personaje en el nido.

- **GameObjectContainer:** Es la clase encargada de almacenar a todos los gameObjects del juego, permite realizar una serie de operaciones con ellos.
- **GameObject:** Clase abstracta que se utiliza para representar un objeto de juego en el servidor. Dicho objeto puede tomar la forma de un jugador, un npc, una criatura o un ítem. Además se encarga de generar el GameObjectInfo que luego será enviado al cliente
- **GameCharacter:** Es la clase que modela a un personaje del lado del servidor. Recibe los inputs enviados por el cliente y los procesa durante el update. Para eso, hace uso de su estado actual. También se encarga de generar el PlayerInfo que luego va a ser enviado al cliente.
- **Creature:** Es un clase que sirve para modelar una criatura del lado del servidor. Al igual que GameCharacter utiliza estados para realizar su update con la diferencia de que no recibe inputs del cliente, sino que los genera por su cuenta permitiéndole alternar su estado actual. La mayor parte del tiempo se encuentra moviéndose sin rumbo dentro de su nido, esperando que un jugador entre en el mismo. Es en ese momento que actualiza su estado para empezar a perseguirlo y eventualmente atacar.
- **NPCServer:** Esta clase modela un NPC del lado del servidor de forma tal que pueda interactuar con el personaje. Todo NPC tiene una profesión que le permite manejar de forma polimórfica la interacción con el personaje. Las profesiones están divididas en 3 variantes, las cuales son:
- **Profession:** Clase abstracta que modela el conocimiento que tendrá un NPCServer. Cada una de sus implementaciones como lo son *Banker*, *Merchant* y *Priest* serán capaz de procesar el input del usuario y brindará un *NPCInfo* con la información correspondiente a esa profesión.
- **ObjectItem:** Hereda de GameObject, su propósito es el de modelar un ítem del lado del servidor. Después de un determinado tiempo y si no fue tomado por algún jugador, desaparece del mapa.
- **State:** Es la clase abstracta de la que dependen tanto los personajes como las criaturas para realizar su update. A pesar que en cliente los estados tienen un comportamiento homólogo, en el servidor no pasa lo mismo. Debido a esto, se procedió a tener ambos tipos de estado por separado, resultando en la creación de 2 clases que heredan de State. Las mismas son:
  - **StatePoolCharacter**
  - **StatePoolCreature**
- **StatePool:** Es la clase abstracta que almacena los posibles estados que puede llegar a tener un GameObject. Al inicializar solo tiene el estado Still y los demás estados se van creando, conforme se necesiten. Tiene un solo estado como el actual y lo va alternando, según los inputs que recibe del jugador o los creados automáticamente por las criaturas. Al tener 2 máquinas de estado distintas, una para el personaje y otra para las criaturas, fue necesario hacer 2 clases que hereden de esta clase.
- **StatePoolCharacter:** Hereda de StatePool, es el encargado de manejar los StatePoolCharacter, que pueden tomar los siguientes valores:
  - **StillStateCharacter**
  - **MoveStateCharacter**
  - **AttackStateCharacter**
  - **EquipStateCharacter**
  - **InteractStateCharacter**
  - **MeditateStateCharacter**
  - **TakeAndDropStateCharacter**
  - **ResurrectStateCharacter**
- **StatePoolCreature:** Hereda de StatePool, es el encargado de manejar los StatePoolCreature, que pueden tomar los siguientes valores:
  - **StillStateCreature**
  - **MoveStateCreature**
  - **AttackStateCreature**
  - **PursuitStateCreature**

4.3. Diagramas UML

El siguiente diagrama nos permite observar como está compuesta la clase *World* y como se relaciona con las clases que la componen. En

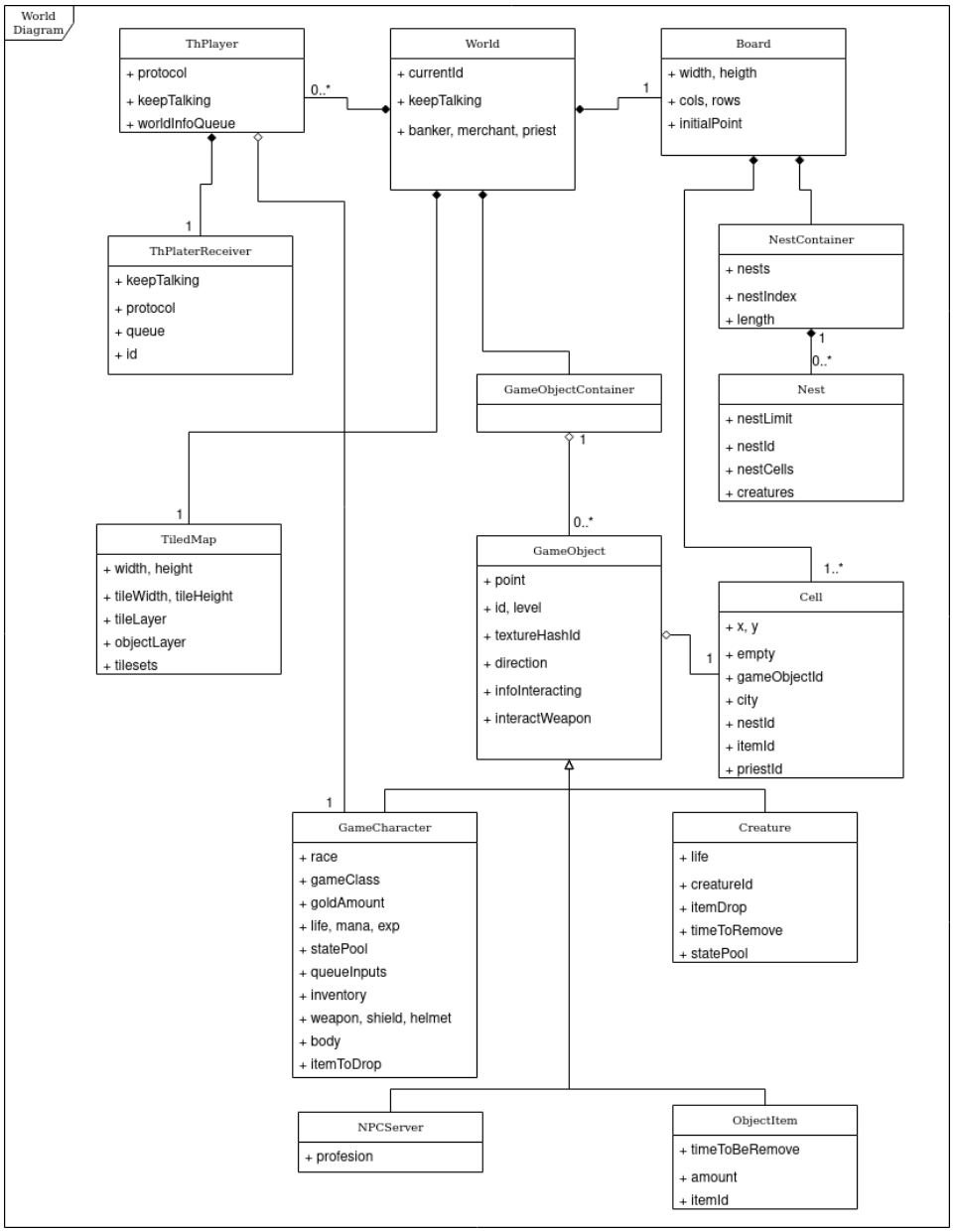

Figura 3: Diagrama Clase World

A continuación, se podrán observar los diagramas de estado para los *Character* y las *Creatures* respectivamente. En ellos se puede observar algunos de los mensajes que realizan que un personaje de Argentum, cambie de estado:

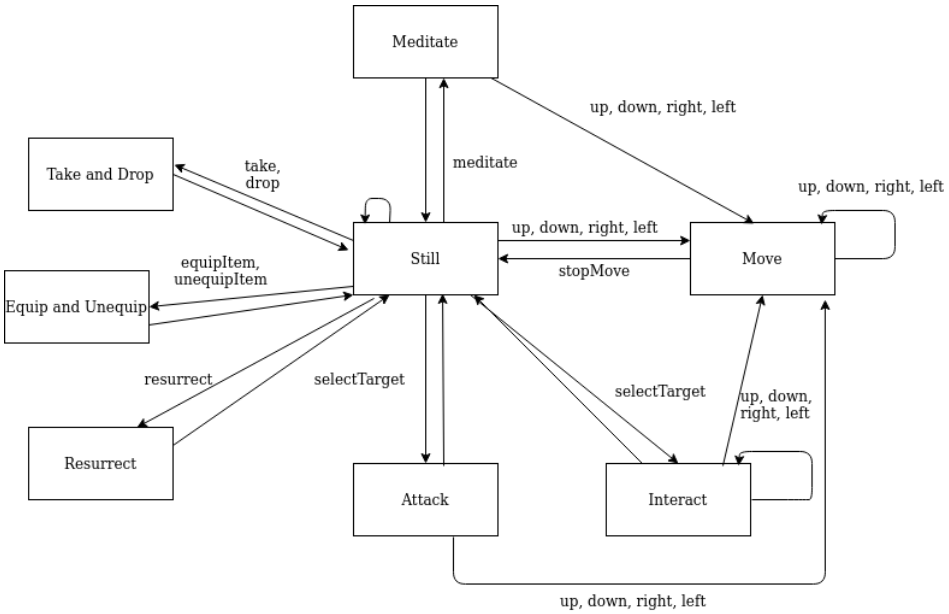

Figura 4: Diagrama Estado Character

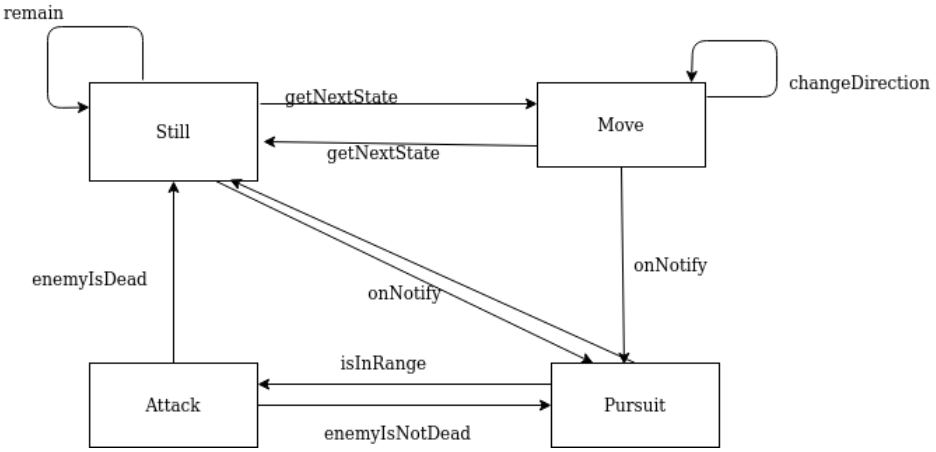

Figura 5: Diagrama Estado Creatures

5. Protocolo de Comunicación

El protocolo de comunicación utilizado fue diseñado a medida de las necesidades del juego. Se trata de un protocolo binario que consta de diferentes tipos de mensajes.

Al entablarse una comunicación entre el cliente y el servidor, el cliente enviará un mensaje de inicio que está compuesto por la raza y la clase seleccionada para su personaje. Una vez recibido esto, el servidor le enviará al cliente el mapa estático y la información inicial para un nuevo jugador creado con las características indicadas en el mensaje de inicio.

Una vez realizado estos 3 pasos, ya el servidor irá enviando cada cierto intervalo de tiempo a todos sus clientes el estado del mundo. Esto incluye el correspondiente PlayerInfo para ese cliente y todos los GameObjects que estén dentro del mundo de Argentum en ese momento.

En caso de que el jugador esté interactuando con alguno de los distintos profesionales dispersos por el mapa, también se enviará un mensaje con la información que este tenga para brindarnos.

Por último, el cliente irá enviando al servidor todos los comandos ingresados por el jugador. Para ello, también hay una codificación especial.

El encargado de generar todo el encoding y decoding de los mensajes es una clase estática llamada *Decoder*. Cada método de encoding recibirá el tipo de dato a convertir y devolverá un vector de bytes. Por otro lado, cada método de decoding recibirá un *Message* y devolverá el tipo de dato ya creado. A continuación se explicará los diversos mensajes y la forma de entender cada uno.

Antes de cada mensaje, se envía la longitud que tendrá el mismo en un entero sin signo de 4 bytes en big endian. Seguido de estos 4 bytes, viene un byte que indica el tipo de mensaje que es. Por último viene el mensaje. Todos los números están en formato big endian.

5.1. Mapa

Si el primer byte del mensaje es un 0, estamos frente al mapa estático del juego. En el cuál se le notifica al cliente la estructura del mapa estático. Para ello, el mensaje se encodea de la siguiente manera:

- **Ancho Mapa:** entero de 2 bytes sin signo en big endian.
- **Alto Mapa:** entero de 2 bytes sin signo en big endian.
- **Ancho Tile:** 1 byte indicando el ancho del tile.
- **Alto Tile:** 1 byte indicando el alto del tile.
- **Cantidad de Layers:** 1 byte indicando la cantidad de layers a enviar.
- **Cantidad Tiles en Layer:** entero de 2 bytes sin signo en big endian, que indica la cantidad de tiles en cada layer.
- A continuacion se deberá enviar Cantidad de Layers * Cantidad Tiles en Layer, la información de cada tile codificada de la siguiente manera :
- **Información de Tile:** entero de 2 bytes sin signo en big endian.
- Comienzo de codificación de TileSets:
- **Cantidad de TileSets:** entero de 4 bytes sin signo en big endian.
- Por cada tileSet especificado en Cantidad de TileSets, se deberá codificar lo siguiente:
- **Primer GID TileSet:** entero de 2 bytes sin signo en big endian.
- **TileSet ID:** entero de 2 bytes sin signo en big endian.

5.2. PlayerInfo

Si el primer byte del mensaje es un 1, estamos frente a un tipo de mensaje de PlayerInfo. En el cuál, se lo notifica al cliente del estado actual de su jugador. Para ello, el mensaje se encodea de la siguiente manera:

- **ID:** entero de 2 bytes sin signo en big endian.
- **Vida:** entero de 2 bytes sin signo en big endian.
- **Vida Máxima:** entero de 2 bytes sin signo en big endian.
- **Mana:** entero de 2 bytes sin signo en big endian.
- **Mana Máximo:** entero de 2 bytes sin signo en big endian.
- **Experiencia:** entero de 4 bytes sin signo en big endian.
- **Experiencia Máxima:** entero de 4 bytes sin signo en big endian.
- **Nivel:** entero de 2 bytes sin signo en big endian.
- **Oro:** entero de 2 bytes sin signo en big endian.
- **Oro seguro:** entero de 2 bytes sin signo en big endian.
- **Equipamiento:** 5 bytes que representan al HelmetID, HeadID, BodyID, ShieldID y WeaponID. Siempre deben estar en el mismo orden, en caso de que no tenga equipado nada, irá un byte en 0.
- **Inventario:** 9 bytes que representan a los ItemsInventoryID que están dentro del inventario del jugador. En caso de que la posición esté vacía ira un byte en 0.
- **Estado:** entero de 2 bytes sin signo en big endian. Representa el CharacterStateID del jugador.

- **Dirección:** entero de 2 bytes sin signo en big endian. Representa la Direction del jugador.
- **Posición:** 2 enteros de 2 bytes sin signo en big endian. Representan la posición en x y la posición en y respectivamente.
- **Atacado por:** un byte que representa el WeaponID en caso de que el jugador esté siendo atacado por algún arma en particular. Si no hay ataque, el byte va en 0.

5.3. GameObjects

Si el primer byte del mensaje es un 2, estamos frente a un tipo de mensaje de GameObjects. En el cuál, se lo notifica al cliente el estado de todos los objetos renderizables del mundo. Para ello, el mensaje se encodea de la siguiente manera:

- **Cantidad de Objetos:** un entero sin signo de 4 bytes en big endian. Es la cantidad de objetos que contiene el vector de GameObjectsInfo a decodificar.
- Por cada objeto del mundo se tendrá que codificar lo siguiente y de manera contigua:
- **ID:** entero de 2 bytes sin signo en big endian.
- **Tipo:** un byte que si es 0, se trata de un personaje y si es 1 de un item resultante de algún drop.
- **Equipamiento:** 6 bytes que representan al HelmetID, HeadID, BodyID, ShieldID , WeaponID y ItemsInventoryID Siempre deben estar en el mismo orden, en caso de que no tenga equipado nada, irá un byte en 0.
- **Estado:** entero de 2 bytes sin signo en big endian. Representa el CharacterStateID del jugador. En caso de ser un item, ira en 0.
- **Dirección:** entero de 2 bytes sin signo en big endian. Representa la Direction del jugador. En caso de ser un item, ira en 0.
- **Posición:** 2 enteros de 2 bytes sin signo en big endian. Representan la posición en x y la posición en y respectivamente.
- **Atacado por:** un byte que representa el WeaponID en caso de que el jugador esté siendo atacado por algún arma en particular. Si no hay ataque, el byte va en 0.

5.4. InputInfo

Si el primer byte del mensaje es un 3, estamos en frente a un comando del usuario. Este mensaje va desde el cliente al servidor. Para ello, el mensaje se encodea de la siguiente manera:

- **InputID:** 1 byte indicando el id del input ingresado.
- **Posición:** 2 enteros de 2 bytes sin signo en big endian. Representan la posición en x y la posición en y respectivamente del click realizado por el usuario.
- **Aditional:** entero de 2 bytes sin signo en big endian. Representa una información adicional que puede ser de utilidad para algunos inputs.

5.5. Mensaje de Inicio

Si el primer byte del mensaje es un 4, estamos en frente a un comando del usuario. Este mensaje va desde el cliente al servidor. Para ello, el mensaje se encodea de la siguiente manera:

- **RaceID:** un byte que representa el id de la raza elegida por el jugador.
- **ClassID:** un byte que representa el id de la clase elegida por el jugador.

5.6. Mensaje de NPC

Si el primer byte del mensaje es un 5, estamos frente a un mensaje que contiene un *NPCInfo*, dado que el jugador está interactuando con un profesional. Para ello, el mensaje se encodea de la siguiente manera:

- **Tipo NPC:** un byte que nos indica con que NPC estamos interactuando. Si es un 1 estamos hablando con un Comerciante, si es un 2 con el sacerdote y si es un 3 con el banquero.
- **Cantidad Acciones:** un byte que nos indica la cantidad de acciones que realiza el NPC.
- **Acciones:** un byte que representa el id de cada una de las acciones que realiza el NPC.
- **Cantidad Items:** un byte que nos indica la cantidad de items que tiene disponible ese NPC.
- **Items:** son duplas de un byte y un entero de 2 bytes sin signo en big endian. Representan el id del item junto a su costo en oro.
- **Oro:** un entero de 2 bytes sin signo en big endian que representa la cantidad de oro depositada en la cuenta bancaria. Aplica solamente para el banquero.
- **Cantidad Items Depositados:** un byte que nos indica la cantidad de items que tiene disponible el banquero en la cuenta bancaria.
- **Items en Banco:** un byte por cada item en el banco que representa el ItemsInventoryID del mismo.

6. Código

jul 21, 20 15:20 **WorldInfoQueue.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_WORLDINFOQUEUE_H
#define ARGENTUM_TALLER_WORLDINFOQUEUE_H

#include "WorldInfo.h"
#include "../common/BlockingQueue.h"

using WorldInfoQueue = BlockingQueue<WorldInfo>;

#endif //ARGENTUM_TALLER_WORLDINFOQUEUE_H
```

jul 21, 20 15:20 **WorldInfo.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_WORLDINFO_H
#define ARGENTUM_TALLER_WORLDINFO_H


#include "../common/PlayerInfo.h"
#include "GameObject.h"

class WorldInfo {
private:
    PlayerInfo playerInfo;
    std::vector<GameObjectInfo> gameObjectsInfo;
    NPCInfo npcInfo;
public:
    WorldInfo(std::vector<std::shared_ptr<GameObject>> gameObjects, uint id);

    const PlayerInfo &getPlayerInfo() const;

    const std::vector<GameObjectInfo> &getGameObjectsInfo() const;

    const NPCInfo &getNpcInfo() const;

    virtual ~WorldInfo();

};


#endif //ARGENTUM_TALLER_WORLDINFO_H
```

jul 21, 20 15:20 **WorldInfo.cpp** Page 1/1

```cpp
#include "WorldInfo.h"
#include "GameCharacter.h"

WorldInfo::~WorldInfo() = default;

WorldInfo::WorldInfo(std::vector<std::shared_ptr<GameObject>> gameObjects, uint id) :
    playerInfo(), gameObjectsInfo(), npcInfo(){
    std::vector<std::shared_ptr<GameObject>>::iterator iter;
    iter = gameObjects.begin();
    while (iter ≠ gameObjects.end()){
        if ((*iter)→getId() ≡ id) {
            this→npcInfo = (*iter)→getInteractInfo();
            this→playerInfo = (*iter)→getPlayerInfo();
            iter = gameObjects.erase(iter);
        } else {
            this→gameObjectsInfo.push_back((*iter)→getGameObjectInfo());
            iter++;
        }
    }
}

const PlayerInfo &WorldInfo::getPlayerInfo() const {
    return playerInfo;
}

const std::vector<GameObjectInfo> &WorldInfo::getGameObjectsInfo() const {
    return gameObjectsInfo;
}

const NPCInfo &WorldInfo::getNpcInfo() const {
    return npcInfo;
}
```

jul 21, 20 15:20 **World.h** Page 1/2

```cpp
#ifndef WORLD_H
#define WORLD_H

#include <string>
#include <unordered_map>
#include <mutex>
#include <atomic>
#include "../common/TiledMap.h"
#include "Board.h"
#include "GameObject.h"
#include "GameStatsConfig.h"
#include "../common/ObjectLayer.h"
#include "../common/Thread.h"
#include "../common/InputQueue.h"
#include "GameCharacter.h"
#include "ThPlayer.h"
#include "Banker.h"
#include "Merchant.h"
#include "Priest.h"
#include "GameObjectsContainer.h"


class World: public Thread {
private:
    TiledMap map;
    Board board;
    GameStatsConfig& gameStatsConfig;
    GameObjectsContainer gameObjectsContainer;
    std::atomic<uint> current_id;
    std::atomic<bool> keepTalking;
    mutable std::mutex m;
    std::unordered_map<uint, ThPlayer*> players;
    Banker* banker;
    Merchant* merchant;
    Priest* priest;

    void addNPCs(std::vector<ObjectLayer> objectLayers);

    uint getNextId();

    std::vector<std::shared_ptr<GameObject>> getUpdatedGameObjects();

    void addCreatures();

    void generateCreature();

    void generateItem(const DropItem &dropItem, const std::shared_ptr<Cell>& emptyCell);

    void clearFinishedPlayers();

    void removeCreaturesAndItems();

    void update();

    void checkDrops();
public:
    explicit World(GameStatsConfig& configuration);

    TiledMap& getStaticMap();

    std::shared_ptr<GameCharacter> createCharacter(RaceID race, GameClassID gameClass);

    virtual void run();

```

jul 21, 20 15:20 **World.h** Page 2/2

```
65      void stop();
66
67      void addPlayer(ThPlayer* aPlayer,uint id);
68
69      virtual ~World();
70  };
71
72  #endif
```

jul 21, 20 15:20 **World.cpp** Page 1/3

```
1   #include "World.h"
2   #include "../common/JsonReader.h"
3   #include "NPCServer.h"
4   #include "../common/Random.h"
5   #include "Creature.h"
6   #include "ObjectItem.h"
7   #include <iostream>
8
9   #define GAMELOOPTIME 1000000/45.0
10
11  World::World(GameStatsConfig& configuration) : gameStatsConfig(configuration),
12      current_id(1), keepTalking(true) {
13      std::string path(CONFIG_DIR+std::string("/finishedMap.json"));
14      rapidjson::Document jsonMap = JsonReader::read(path);
15      this→banker = Banker::getInstance();
16      this→merchant = Merchant::getInstance();
17      this→merchant→init(configuration.getItems());
18      this→priest = Priest::getInstance();
19      this→priest→init(configuration.getItems());
20      this→merchant→init(configuration.getItems());
21      this→priest→init(configuration.getItems());
22      this→map = TiledMap(jsonMap);
23      this→board = Board(map, GameStatsConfig::getNestCreatureLimit());
24      addNPCs(map.getObjectLayers());
25      addCreatures();
26  }
27
28  uint World::getNextId() {
29      return current_id++;
30  }
31
32  std::shared_ptr<GameCharacter> World::createCharacter(RaceID race, GameClassID g
    ameClass) {
33      std::unique_lock<std::mutex> lock(m);
34      uint id = getNextId();
35      std::shared_ptr<Cell> initialCell = board.getInitialCell();
36      initialCell→occupied(id);
37      std::shared_ptr<GameCharacter> aCharacter(new GameCharacter(id, race, gameCl
    ass, initialCell, board.getPointFromCell(initialCell)));
38      gameObjectsContainer.addGameObject(aCharacter, id);
39      return aCharacter;
40  }
41
42  void World::addNPCs(std::vector<ObjectLayer> objectLayers) {
43      for (auto &anObjectLayer : objectLayers) {
44          if (anObjectLayer.getName() == "NPC") {
45              std::shared_ptr<Cell> aCell;
46              for (StaticObject &aNPCObject : anObjectLayer.getObjects()) {
47                  uint id = getNextId();
48                  aCell = board.getCellFromPoint(aNPCObject.getTopLeft());
49                  aCell→occupied(id);
50                  std::shared_ptr<NPCServer> aNPC(new NPCServer(id, aNPCObject.get
    Name(), board.getPointFromCell(aCell), aCell));
51                  gameObjectsContainer.addGameObject(aNPC, id);
52              }
53          }
54      }
55  }
56
57  void World::addCreatures() {
58      for (int i = 0; i < GameStatsConfig::getCreaturesLimit(); ++i) {
59          generateCreature();
60      }
61  }
62
63  void World::generateCreature() {
```

jul 21, 20 15:20 **World.cpp** Page 2/3

```
64      uint id = getNextId();
65      uint8_t randomId = Random::get(1, 4);
66      try {
67          Nest& aNest = board.getAvailableNest();
68          std::shared_ptr<Cell> initialCell = board.getInitialCellInNest(aNest);
69          initialCell→occupied(id);
70          aNest.addCreature(id);
71          std::shared_ptr<Creature> aCreature(new Creature(id, CreatureID(randomId
    ), initialCell, board.getPointFromCell(initialCell)));
72          gameObjectsContainer.addGameObject(aCreature, id);
73      } catch (Exception& e) {
74          std::cout << "Cannot create creature" << std::endl;
75      }
76
77  }
78
79  void World::generateItem(const DropItem& dropItem, const std::shared_ptr<Cell>&
    initialCell) {
80      uint id = getNextId();
81      std::shared_ptr<ObjectItem> aObjectItem(new ObjectItem(id, board.getPointFro
    mCell(initialCell), initialCell, dropItem));
82      initialCell→setItemId(id);
83      gameObjectsContainer.addGameObject(aObjectItem, id);
84      std::cout << "Item created -- ID: " << id << std::endl;
85  }
86
87
88  void World::run() {
89      Chrono chrono;
90      double initLoop, endLoop, sleep;
91      int amountCreaturesDiff;
92      std::vector<std::shared_ptr<GameObject>> gameObjects;
93      while (keepTalking) {
94          initLoop = chrono.lap();
95          amountCreaturesDiff = GameStatsConfig::getCreaturesLimit() - board.getAm
    ountCreatures();
96          if (amountCreaturesDiff > 0) {
97              generateCreature();
98          }
99          update();
100         clearFinishedPlayers();
101         for (auto &aPlayer : players) {
102             gameObjects = gameObjectsContainer.getUpdatedGameObjects();
103             aPlayer.second→update(WorldInfo(gameObjects, aPlayer.first));
104         }
105         checkDrops();
106         removeCreaturesAndItems();
107         endLoop = chrono.lap();
108         sleep = GAMELOOPTIME - (endLoop - initLoop);
109         if (sleep > 0)
110             usleep(sleep);
111     }
112 }
113
114 TiledMap& World::getStaticMap() {
115     return this→map;
116 }
117
118 void World::stop() {
119     this→keepTalking = false;
120     for (auto & player : this→players) {
121     player.second→stop();
122         player.second→join();
123         delete player.second;
124     }
125 }
```

jul 21, 20 15:20 **World.cpp** Page 3/3

```
126
127 void World::update() {
128     gameObjectsContainer.update(board);
129 }
130
131 std::vector<std::shared_ptr<GameObject>> World::getUpdatedGameObjects() {
132     return gameObjectsContainer.getUpdatedGameObjects();
133 }
134
135 void World::addPlayer(ThPlayer *aPlayer,uint id) {
136     aPlayer→start();
137     players.insert({id,aPlayer});
138 }
139
140 void World::clearFinishedPlayers() {
141     auto iter = this→players.begin();
142     while (iter ≠ this→players.end()) {
143         if (¬(*iter).second→is_alive()) {
144             (*iter).second→join();
145             gameObjectsContainer.deleteGameObject((*iter).first, board);
146             delete (*iter).second;
147             iter = this→players.erase(iter);
148         } else {
149             iter++;
150         }
151     }
152 }
153
154 void World::removeCreaturesAndItems() {
155     gameObjectsContainer.removeCreaturesAndItems(board);
156 }
157
158 void World::checkDrops() {
159     for (auto &aGameObject : gameObjectsContainer.getUpdatedGameObjects()) {
160         if (aGameObject→canDropsItems()) {
161             for (auto &dropItem : aGameObject→getDrop()) {
162                 generateItem(dropItem, board.getNextEmptyCell(aGameObject→getAc
    tualCell()));
163             }
164         }
165     }
166 }
167
168 World::~World() {
169     delete priest;
170     delete merchant;
171     delete banker;
172 }
173
```

jul 21, 20 15:20 **ThPlayerReceiver.h** Page 1/1

```h
#ifndef THPLAYERRECEIVER_H
#define THPLAYERRECEIVER_H

#include <atomic>
#include "../common/Thread.h"
#include "../common/CommunicationProtocol.h"
#include "../common/Decoder.h"
#include "../common/InputQueue.h"

class ThPlayerReceiver: public Thread {
private:
    std::atomic<bool> keepTalking;
    std::shared_ptr<CommunicationProtocol> protocol;
    InputQueue& queue;
    uint id{0};

public:
    explicit ThPlayerReceiver(std::shared_ptr<CommunicationProtocol> protocol, InputQueue& queue);

    virtual void run();

    void stop();

    void setId(uint id);

    virtual ~ThPlayerReceiver();
};

#endif
```

jul 21, 20 15:20 **ThPlayerReceiver.cpp** Page 1/1

```cpp
#include "ThPlayerReceiver.h"
#include <iostream>
#include <utility>
#include "../common/SocketException.h"

#define UNKNOW_ERROR "Unknow Error"
#define ERRORSOCKET "Error en la comunicaciÃ³n en ThPlayerReceiver::run() "
#define ERRORRECEIVER "Error en ThPlayerReceiver::run() "


ThPlayerReceiver::ThPlayerReceiver(std::shared_ptr<CommunicationProtocol> protocol, InputQueue& queue) :
    keepTalking(true), protocol(std::move(protocol)), queue(queue) {}

void ThPlayerReceiver::setId(uint id) {
    this->id = id;
}

void ThPlayerReceiver::run() {
    Message msg;
    InputInfo input;
    while(this->keepTalking) {
        try{
            msg = this->protocol->receive();
            input = Decoder::decodeCommand(msg);
            this->queue.push(std::move(input));
        } catch (const SocketException& e) {
            std::cerr << ERRORSOCKET << e.what() << std::endl;
            this->keepTalking = false;
        } catch(const std::exception& e){
            std::cerr << ERRORRECEIVER << e.what() << std::endl;
            this->keepTalking = false;
        }catch (...) {
            this->keepTalking = false;
            std::cerr << UNKNOW_ERROR << std::endl;
        }
    }
}

void ThPlayerReceiver::stop() {
    this->keepTalking = false;
}

ThPlayerReceiver::~ThPlayerReceiver() = default;
```

jul 21, 20 15:20 **ThPlayer.h** Page 1/1

```h
#ifndef SERVERPLAYER_H
#define SERVERPLAYER_H

#include <atomic>
#include <mutex>
#include <condition_variable>
#include "../common/Thread.h"
#include "../common/CommunicationProtocol.h"
#include "../common/Socket.h"
#include "ThPlayerReceiver.h"
#include "../common/PlayerInfo.h"
#include "GameCharacter.h"
#include "WorldInfoQueue.h"


class ThPlayer : public Thread {
private:
    std::shared_ptr<CommunicationProtocol> protocol;
    std::atomic<bool> keepTalking;
    std::shared_ptr<GameCharacter> character;
    ThPlayerReceiver receiver;
    WorldInfoQueue worldInfoQueue;

public:
    ThPlayer(const std::shared_ptr<CommunicationProtocol>& protocol, std::shared_ptr<GameCharacter> aCharacter);

    virtual void run();

    void stop();

    bool is_alive() const;

    void update(const WorldInfo& worldInfo);

    virtual ~ThPlayer();
};


#endif
```

jul 21, 20 15:20 **ThPlayer.cpp** Page 1/1

```cpp
#include "ThPlayer.h"
#include <iostream>
#include <utility>
#include "../common/SocketException.h"

#define UNKNOW_ERROR "Unknow Error"
#define ERRORSOCKET "Error en la comunicaciÃ³nn en ThPlayer::run() "
#define ERRORDISPATCHER "Error en ThPlayer::run() "

ThPlayer::ThPlayer(const std::shared_ptr<CommunicationProtocol>& protocol, std::shared_ptr<GameCharacter> aCharacter) :
protocol(protocol), keepTalking(true), character(std::move(aCharacter)),
receiver(protocol, character→getQueueInputs()), worldInfoQueue(true) {}

void ThPlayer::run() {
    this→receiver.setId(character→getId());
    this→receiver.start();
    while (this→keepTalking) {
        try{
            WorldInfo worldInfo = worldInfoQueue.pop();
            const PlayerInfo& aPlayerInfo = worldInfo.getPlayerInfo();
            this→protocol→send(Decoder::encodePlayerInfo(aPlayerInfo));
            this→protocol→send(Decoder::encodeGameObjects(worldInfo.getGameObjectsInfo()));
            if (aPlayerInfo.getState() ≡ CharacterStateID::Interact ∧ worldInfo.getNpcInfo().type ≠ 0) {
                this→protocol→send(Decoder::encodeNPCInfo(worldInfo.getNpcInfo()));
            }

        } catch(const SocketException& e) {
            std::cout << ERRORSOCKET << e.what() << std::endl;
            this→stop();
        } catch(const std::exception& e){
            std::cerr << ERRORDISPATCHER << e.what() << std::endl;
            this→stop();
        } catch (...) {
            std::cerr << UNKNOW_ERROR << std::endl;
            this→stop();
        }
    }
}

void ThPlayer::stop() {
    this→keepTalking = false;
    this→receiver.stop();
    this→receiver.join();
    this→protocol→stop();
}

bool ThPlayer::is_alive() const {
    return this→keepTalking;
}

void ThPlayer::update(const WorldInfo& worldInfo) {
    worldInfoQueue.push(worldInfo);
}

ThPlayer::~ThPlayer()= default;
```

jul 21, 20 15:20 **ThLobbyPlayer.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_THLOBBYPLAYER_H
#define ARGENTUM_TALLER_THLOBBYPLAYER_H


#include "../common/Thread.h"
#include "World.h"
#include "../common/Socket.h"
#include "../common/CommunicationProtocol.h"

class ThLobbyPlayer : public Thread {
private:
    std::atomic<bool> keepTalking;
    World& world;
    std::shared_ptr<CommunicationProtocol> protocol;
public:
    ThLobbyPlayer(Socket socket, World& world);

    void run() override;

    void stop();

    bool is_alive() const;

    ~ThLobbyPlayer() override;
};


#endif //ARGENTUM_TALLER_THLOBBYPLAYER_H
```

jul 21, 20 15:20 **ThLobbyPlayer.cpp** Page 1/1

```cpp
#include <iostream>
#include <memory>
#include "ThLobbyPlayer.h"

#define INITMSG 0x04

void ThLobbyPlayer::run() {
    Message welcomeMsg = this→protocol→receive();
    std::shared_ptr<GameCharacter> aCharacter;
    if (welcomeMsg.getType() ≡ INITMSG) {
        RaceID race = (RaceID) welcomeMsg.read8();
        GameClassID gameClass = (GameClassID) welcomeMsg.read8();
        aCharacter = world.createCharacter(race, gameClass);
    }
    std::vector<uint8_t> map = Decoder::encodeMap(this→world.getStaticMap());
    this→protocol→send(map);
    this→protocol→send(Decoder::encodePlayerInfo(aCharacter→getPlayerInfo()));
    world.addPlayer(new ThPlayer(protocol, aCharacter),aCharacter→getId());
    keepTalking = false;
}

ThLobbyPlayer::~ThLobbyPlayer() = default;

ThLobbyPlayer::ThLobbyPlayer(Socket socket, World &world) :
keepTalking(true), world(world) {
    protocol = std::make_shared<CommunicationProtocol>(std::move(socket));
}

void ThLobbyPlayer::stop() {
    this→keepTalking = false;
    this→protocol→stop();
}

bool ThLobbyPlayer::is_alive() const {
    return keepTalking;
}
```

jul 21, 20 15:20 **TakeAndDropStateCharacter.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_TAKEANDDROPSTATECHARACTER_H
#define ARGENTUM_TALLER_TAKEANDDROPSTATECHARACTER_H


#include "StateCharacter.h"

class TakeAndDropStateCharacter : public StateCharacter {
public:
    explicit TakeAndDropStateCharacter();

    ~TakeAndDropStateCharacter() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    bool isAttacking() override;

    void init(InputInfo aInputInfo) override;

    bool isMeditating() override;
};


#endif //ARGENTUM_TALLER_TAKEANDDROPSTATECHARACTER_H
```

jul 21, 20 15:20 **TakeAndDropStateCharacter.cpp** Page 1/1

```
#include "TakeAndDropStateCharacter.h"
#include "../GameCharacter.h"
#include "../ObjectItem.h"

TakeAndDropStateCharacter::~TakeAndDropStateCharacter() {
    stateId = StateID::TakeDrop;
}

TakeAndDropStateCharacter::TakeAndDropStateCharacter() : StateCharacter() {}

void TakeAndDropStateCharacter::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects,
                                            Board &board) {

    std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(id));
    std::shared_ptr<Cell> characterCell = aCharacter→getActualCell();
    switch (inputInfo.input) {
        case InputID::dropItem:
            if (¬characterCell→hasItem()) {
                aCharacter→dropItem(inputInfo.aditional − 1);
            }
            break;
        case InputID::takeItem:
            if (characterCell→hasItem()) {
                std::shared_ptr<ObjectItem> anItem = std::dynamic_pointer_cast<ObjectItem>(gameObjects.at(characterCell→getItemId()));
                if(aCharacter→takeItem(anItem→getItemId(), anItem→getAmount())) {
                    anItem→take();
                }
            }
            break;
        default:
            break;
    }
    finalized = true;
}

StateID TakeAndDropStateCharacter::getNextStateID(InputInfo info) {
    StateID nextState = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextState = StateID::Move;
    }
    return nextState;
}

StateID TakeAndDropStateCharacter::getResetStateID() {
    return StateID::Still;
}

bool TakeAndDropStateCharacter::isAttacking() {
    return false;
}

bool TakeAndDropStateCharacter::isMeditating() {
    return false;
}

void TakeAndDropStateCharacter::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
}
```

jul 21, 20 15:20 **StillStateCreature.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_STILLSTATECREATURE_H
#define ARGENTUM_TALLER_STILLSTATECREATURE_H

#include "State.h"
#include "StateCreature.h"

class StillStateCreature : public StateCreature {
private:
public:
    StillStateCreature();

    ~StillStateCreature() override;

    bool isOnPursuit(uint pursuitId) override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    bool isAttacking() override;

    StateID getResetStateID() override;

    void init(InputInfo aInputInfo) override;
};


#endif //ARGENTUM_TALLER_STILLSTATECREATURE_H
```

jul 21, 20 15:20 **StillStateCreature.cpp** Page 1/1

```cpp
#include "StillStateCreature.h"

StillStateCreature::StillStateCreature() : StateCreature() {
    finalized = true;
    stateId = StateID::Still;
}

void StillStateCreature::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects,
                                     Board &board) {}

StateID StillStateCreature::getNextStateID(InputInfo info) {
    return StateID::Move;
}

bool StillStateCreature::isOnPursuit(uint pursuitId) {
    return false;
}

bool StillStateCreature::isAttacking() {
    return false;
}

void StillStateCreature::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
}

StateID StillStateCreature::getResetStateID() {
    return StateCreature::getResetStateID();
}

StillStateCreature::~StillStateCreature() = default;
```

jul 21, 20 15:20 **StillStateCharacter.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_STILLSTATECHARACTER_H
#define ARGENTUM_TALLER_STILLSTATECHARACTER_H

#include "StateCharacter.h"

class StillStateCharacter: public StateCharacter {
public:
    StillStateCharacter();

    ~StillStateCharacter() override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    void init(InputInfo aInputInfo) override;

    bool isMeditating() override;

    bool isAttacking() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

};


#endif //ARGENTUM_TALLER_STILLSTATECHARACTER_H
```

jul 21, 20 15:20 **StillStateCharacter.cpp** Page 1/1

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

StillStateCharacter::~StillStateCharacter() = default;

StillStateCharacter::StillStateCharacter() : StateCharacter() {
    finalized = true;
    stateId = StateID::Still;
}

void StillStateCharacter::performTask(uint id,
        std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {}

StateID StillStateCharacter::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextStateId = StateID::Move;
    } else if (info.input ≡ InputID::selectTarget) {
        nextStateId = StateID::Transition;
    } else if (info.input ≡ InputID::equipItem ∨ info.input ≡ InputID::unequipItem) {
        nextStateId = StateID::Equip;
    } else if (info.input ≡ InputID::takeItem ∨ info.input ≡ InputID::dropItem) {
        nextStateId = StateID::TakeDrop;
    } else if (info.input ≡ InputID::meditate) {
        nextStateId = StateID::Meditate;
    } else if (info.input ≡ InputID::resurrect) {
        nextStateId = StateID::Resurrect;
    }
    return nextStateId;
}

StateID StillStateCharacter::getResetStateID() {
    return StateID::Still;
}

bool StillStateCharacter::isAttacking() {
    return false;
}

bool StillStateCharacter::isMeditating() {
    return false;
}

void StillStateCharacter::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
}

```

jul 21, 20 15:20 **StateTranslator.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_STATETRANSLATOR_H
#define ARGENTUM_TALLER_STATETRANSLATOR_H


#include "../../common/Identificators.h"

class StateTranslator {
public:
    StateTranslator();

    static CharacterStateID stateToCharacterState(StateID stateId);

    virtual ~StateTranslator();
};


#endif //ARGENTUM_TALLER_STATETRANSLATOR_H
```

jul 21, 20 15:20 **StateTranslator.cpp** Page 1/1

```cpp
#include "StateTranslator.h"

StateTranslator::StateTranslator() = default;

StateTranslator::~StateTranslator() = default;

CharacterStateID StateTranslator::stateToCharacterState(StateID stateId) {
    CharacterStateID characterStateId = CharacterStateID::Still;
    switch (stateId) {
        case StateID::Still:
            characterStateId = CharacterStateID::Still;
            break;
        case StateID::Move:
            characterStateId = CharacterStateID::Move;
            break;
        case StateID::Equip:
            characterStateId = CharacterStateID::Still;
            break;
        case StateID::Transition:
            characterStateId = CharacterStateID::Still;
            break;
        case StateID::Interact:
            characterStateId = CharacterStateID::Interact;
            break;
        case StateID::Attack:
            characterStateId = CharacterStateID::Attack;
            break;
        case StateID::Pursuit:
            characterStateId = CharacterStateID::Move;
            break;
        case StateID::Meditate:
            characterStateId = CharacterStateID::Meditate;
            break;
        case StateID::TakeDrop:
            characterStateId =  CharacterStateID::Still;
            break;
        case StateID::Resurrect:
            characterStateId = CharacterStateID::Resurrect;
            break;
    }
    return characterStateId;
}
```

jul 21, 20 15:20 **StatePool.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_STATEPOOL_H
#define ARGENTUM_TALLER_STATEPOOL_H

#include <GameObject.h>
#include "State.h"

class StatePool {
public:
    StatePool();

    virtual void changeState(StateID id, InputInfo aInputInfo) = 0;

    virtual void setNextState(StateID id, InputInfo aInputInfo) = 0;

    virtual void updateState() = 0;

    virtual void performTask(std::unordered_map<uint, std::shared_ptr<GameObject
>> &gameObjects,
             Board &board) = 0;

    virtual StateID getStateId() = 0;

    virtual ~StatePool();
};

#endif //ARGENTUM_TALLER_STATEPOOL_H
```

jul 21, 20 15:20 **StatePoolCreature.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_STATEPOOLCREATURE_H
#define ARGENTUM_TALLER_STATEPOOLCREATURE_H


#include "StatePool.h"
#include "StateCreature.h"

class StatePoolCreature : public StatePool {
private:
    GameObject& creature;
    std::shared_ptr<StateCreature> actualState;
    std::unordered_map<StateID, std::shared_ptr<StateCreature>, std::hash<StateI
D>> states;
public:
    explicit StatePoolCreature(GameObject &creature);

    void updateState() override;

    bool startChasing(uint pursuitId);

    std::shared_ptr<StateCreature> generateState(StateID stateId);

    ~StatePoolCreature() override;

    void changeState(StateID id, InputInfo aInputInfo) override;

    void setNextState(StateID id, InputInfo aInputInfo) override;

    StateID getStateId() override;

    void performTask(std::unordered_map<uint, std::shared_ptr<GameObject>> &game
Objects, Board &board) override;

};


#endif //ARGENTUM_TALLER_STATEPOOLCREATURE_H
```

jul 21, 20 15:20 **StatePoolCreature.cpp** Page 1/2

```cpp
#include "StatePoolCreature.h"
#include "StillStateCreature.h"
#include "AttackStateCreature.h"
#include "MoveStateCreature.h"
#include "PursuitStateCreature.h"

StatePoolCreature::StatePoolCreature(GameObject &aCreature) : creature(aCreature),
actualState(std::shared_ptr<StateCreature>(new StillStateCreature())) {
    states.insert(std::pair<StateID,
            std::shared_ptr<StateCreature>>(actualState→getStateId(), actualState));
}

void StatePoolCreature::updateState() {
    if (actualState→isOver()) {
        InputInfo aInputInfo;
        if (actualState→getStateId() ≡ StateID::Pursuit ∨ actualState→getStateId() ≡ StateID::Attack) {
            aInputInfo = actualState→getInputInfo();
        } else {
            aInputInfo = creature.getNextInputInfo();
        }
        StateID nextStateId = actualState→getNextStateID(aInputInfo);
        setNextState(nextStateId, aInputInfo);
    }
}

void StatePoolCreature::setNextState(StateID id, InputInfo aInputInfo) {
    changeState(id, aInputInfo);
}

void StatePoolCreature::changeState(StateID id, InputInfo aInputInfo) {
    std::shared_ptr<StateCreature> nextState;
    try {
        nextState = states.at(id);
    } catch (std::exception &e) {
        nextState = generateState(id);
    }
    nextState→init(aInputInfo);
    actualState = nextState;
}

std::shared_ptr<StateCreature> StatePoolCreature::generateState(StateID stateId) {
    std::shared_ptr<StateCreature> newState;
    switch (stateId) {
        case StateID::Move:
            newState = std::shared_ptr<StateCreature>(new MoveStateCreature());
            break;
        case StateID::Attack:
            newState = std::shared_ptr<StateCreature>(new AttackStateCreature());
            break;
        case StateID::Pursuit:
            newState = std::shared_ptr<StateCreature>(new PursuitStateCreature());
            break;
        default:
            break;
    }
    states.insert(std::pair<StateID,
            std::shared_ptr<StateCreature>>(newState→getStateId(), newState));
    return newState;
}

```

jul 21, 20 15:20 **StatePoolCreature.cpp** Page 2/2

```cpp
bool StatePoolCreature::startChasing(uint pursuitId) {
    return ¬actualState→isAttacking() ∧ ¬actualState→isOnPursuit(pursuitId);
}

StateID StatePoolCreature::getStateId() {
    return actualState→getStateId();
}

void StatePoolCreature::performTask(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {
    actualState→performTask(creature.getId(), gameObjects, board);
}

StatePoolCreature::~StatePoolCreature() = default;
```

jul 21, 20 15:20 **StatePool.cpp** Page 1/1

```cpp
#include "StatePool.h"

StatePool::~StatePool() = default;

StatePool::StatePool() = default;
```

jul 21, 20 15:20 **StatePoolCharacter.h** Page 1/1

```cpp
#ifndef ARGENTUM_TALLER_STATEPOOLCHARACTER_H
#define ARGENTUM_TALLER_STATEPOOLCHARACTER_H

#include "StatePool.h"
#include "StateCharacter.h"

class StatePoolCharacter : public StatePool {
private:
    GameObject& character;
    std::shared_ptr<StateCharacter> actualState;
    std::unordered_map<StateID, std::shared_ptr<StateCharacter>, std::hash<StateID>> states;
public:
    explicit StatePoolCharacter(GameObject &aCharacter);

    void updateState() override;

    ~StatePoolCharacter() override;

    void performTask(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    bool isPossibleDeadState(StateID id);

    bool isMeditating();

    void changeState(StateID id, InputInfo aInputInfo) override;

    void setNextState(StateID id, InputInfo aInputInfo) override;

    StateID getStateId() override;

    std::shared_ptr<StateCharacter> generateState(StateID id);
};


#endif //ARGENTUM_TALLER_STATEPOOLCHARACTER_H
```

```cpp
#include "StatePoolCharacter.h"
#include "StillStateCharacter.h"
#include "MoveStateCharacter.h"
#include "EquipStateCharacter.h"
#include "InteractStateCharacter.h"
#include "AttackStateCharacter.h"
#include "MeditateStateCharacter.h"
#include "TakeAndDropStateCharacter.h"
#include "ResurrectStateCharacter.h"

StatePoolCharacter::StatePoolCharacter(GameObject &aCharacter) : character(aCharacter),
                                                                 actualState(std::shared_ptr<StateCharacter>(new StillStateCharacter())) {
    states.insert(std::pair<StateID,
            std::shared_ptr<StateCharacter>>(actualState→getStateId(), actualState));
}

void StatePoolCharacter::updateState() {
    if (actualState→isOver()) {
        StateID nextStateId;
        InputInfo aInputInfo = actualState→getInputInfo();
        if (character.hasAnInputInfo()) {
            aInputInfo = character.getNextInputInfo();
            nextStateId = actualState→getNextStateID(aInputInfo);
        } else {
            nextStateId = actualState→getResetStateID();
        }
        setNextState(nextStateId, aInputInfo);
    }
}

void StatePoolCharacter::setNextState(StateID id, InputInfo aInputInfo) {
    if (id ≡ StateID::Transition) {
        id = character.getActualCell()→isCity() ? StateID::Interact : StateID::Attack;
    }
    if (character.isDead() ∧ ¬isPossibleDeadState(id)) {
        return;
    }
    changeState(id, aInputInfo);
}

bool StatePoolCharacter::isPossibleDeadState(StateID id) {
    return id ≡ StateID::Still ∨ id ≡ StateID::Move ∨ id ≡ StateID::Interact ∨ id ≡ StateID::Resurrect;
}

void StatePoolCharacter::changeState(StateID id, InputInfo aInputInfo) {
    std::shared_ptr<StateCharacter> nextState;
    try {
        nextState = states.at(id);
    } catch (std::exception &e) {
        nextState = generateState(id);
    }
    nextState→init(aInputInfo);
    actualState = nextState;
}

std::shared_ptr<StateCharacter> StatePoolCharacter::generateState(StateID stateId) {
    std::shared_ptr<StateCharacter> newState;
    switch (stateId) {
        case StateID::Move:
            newState = std::shared_ptr<StateCharacter>(new MoveStateCharacter());
```



```cpp
            break;
        case StateID::Equip:
            newState = std::shared_ptr<StateCharacter>(new EquipStateCharacter());
            break;
        case StateID::Interact:
            newState = std::shared_ptr<StateCharacter>(new InteractStateCharacter());
            break;
        case StateID::Attack:
            newState = std::shared_ptr<StateCharacter>(new AttackStateCharacter());
            break;
        case StateID::Meditate:
            newState = std::shared_ptr<StateCharacter>(new MeditateStateCharacter());
            break;
        case StateID::TakeDrop:
            newState = std::shared_ptr<StateCharacter>(new TakeAndDropStateCharacter());
            break;
        case StateID::Resurrect:
            newState = std::shared_ptr<StateCharacter>(new ResurrectStateCharacter());
            break;
        default:
            break;
    }
    states.insert(std::pair<StateID,
            std::shared_ptr<StateCharacter>>(newState→getStateId(), newState));
    return newState;
}

StateID StatePoolCharacter::getStateId() {
    return actualState→getStateId();
}

void StatePoolCharacter::performTask(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {
    actualState→performTask(character.getId(), gameObjects, board);
}

bool StatePoolCharacter::isMeditating() {
    return actualState→isMeditating();
}

StatePoolCharacter::~StatePoolCharacter() = default;
```

jul 21, 20 15:20 **State.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_STATE_H
#define ARGENTUM_TALLER_STATE_H

#include <GameObject.h>
#include "../../common/Identificators.h"

class State {
protected:
    bool finalized = false;
    InputInfo inputInfo;
    StateID stateId;
public:
    State();

    virtual ~State();

    StateID getStateId() const;

    const InputInfo &getInputInfo() const;

    bool isOver() const;

    virtual void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) = 0;

    virtual StateID getNextStateID(InputInfo aInputInfo) = 0;

    virtual StateID getResetStateID() = 0;

    virtual bool isAttacking() = 0;

    virtual void init(InputInfo aInputInfo) = 0;
};


#endif //ARGENTUM_TALLER_STATE_H
```

jul 21, 20 15:20 **StateCreature.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_STATECREATURE_H
#define ARGENTUM_TALLER_STATECREATURE_H

#include "State.h"

class StateCreature : public State {
public:
    ~StateCreature() override;

    StateID getResetStateID() override;

    virtual bool isOnPursuit(uint pursuitId) = 0;
};


#endif //ARGENTUM_TALLER_STATECREATURE_H
```

jul 21, 20 15:20 **StateCreature.cpp** Page 1/1

```cpp
#include "StateCreature.h"

StateCreature::~StateCreature() = default;

StateID StateCreature::getResetStateID() {
    return StateID::Still;
}
```

jul 21, 20 15:20 **State.cpp** Page 1/1

```cpp
#include "State.h"

State::State() {
    InputInfo info;
    info.input = InputID::nothing;
    inputInfo = info;
    stateId = StateID::Still;
}

State::~State() = default;

bool State::isOver() const {
    return finalized;
}

StateID State::getStateId() const {
    return stateId;
}

const InputInfo &State::getInputInfo() const {
    return inputInfo;
}
```

jul 21, 20 15:20 **StateCharacter.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_STATECHARACTER_H
#define ARGENTUM_TALLER_STATECHARACTER_H

#include "State.h"

class StateCharacter : public State {
public:
    StateCharacter();

    ~StateCharacter() override;

    virtual bool isMeditating() = 0;
};


#endif //ARGENTUM_TALLER_STATECHARACTER_H
```

jul 21, 20 15:20 **StateCharacter.cpp** Page 1/1

```cpp
#include "StateCharacter.h"

StateCharacter::~StateCharacter() = default;

StateCharacter::StateCharacter() = default;

```

jul 21, 20 15:20 **ResurrectStateCharacter.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_RESURRECTSTATECHARACTER_H
#define ARGENTUM_TALLER_RESURRECTSTATECHARACTER_H

#include <MovementCharacter.h>
#include "StateCharacter.h"

class ResurrectStateCharacter : public StateCharacter {
private:
    std::shared_ptr<Cell> aPriestCell = nullptr;
    MovementCharacter movement;
public:
    ResurrectStateCharacter();

    ~ResurrectStateCharacter() override;

    void
    performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    void init(InputInfo aInputInfo) override;

    bool isAttacking() override;

    bool isMeditating() override;
};


#endif //ARGENTUM_TALLER_RESURRECTSTATECHARACTER_H
```

jul 21, 20 15:20 **ResurrectStateCharacter.cpp** Page 1/2

```cpp
#include <GameCharacter.h>
#include "ResurrectStateCharacter.h"

ResurrectStateCharacter::~ResurrectStateCharacter() = default;

ResurrectStateCharacter::ResurrectStateCharacter() : StateCharacter() {
    finalized = false;
    stateId = StateID::Resurrect;
}

void ResurrectStateCharacter::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {

    std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(id));
    std::shared_ptr<Cell> characterCell = aCharacter→getActualCell();
    if (¬aCharacter→isDead()) {
        finalized = true;
        return;
    }
    if (aPriestCell ≡ nullptr) {
        aPriestCell = board.getCloserPriest(characterCell);
    }
    if (¬movement.hasStart()) {
            if (board.getDistance(characterCell, aPriestCell) ≡ 2) {
                finalized = true;
                aCharacter→setDirection(board.getDirection(characterCell, aPriestCell));
                aCharacter→cure();
            } else {
                std::shared_ptr<Cell> newCell;
                newCell = board.getBestCell(characterCell, aPriestCell, true);
                if (newCell ≠ characterCell) {
                    Direction aDirection = board.getDirection(characterCell, newCell);
                    aCharacter→setDirection(aDirection);
                    movement.start(board.getPointFromCell(aCharacter→getActualCell()), aDirection, aCharacter→getRace());
                    aCharacter→getActualCell()→free();
                    newCell→occupied(id);
                    aCharacter→setCell(newCell);
                }
            }
    } else {
        aCharacter→setPoint(movement.doStep());
        if (movement.isOver()) {
            movement.reset();
        }
    }

}

StateID ResurrectStateCharacter::getNextStateID(InputInfo info) {
    return StateID::Still;
}

StateID ResurrectStateCharacter::getResetStateID() {
    return StateID::Still;
}

bool ResurrectStateCharacter::isAttacking() {
    return false;
}

bool ResurrectStateCharacter::isMeditating() {
    return false;
```

jul 21, 20 15:20 **ResurrectStateCharacter.cpp** Page 2/2

```
}

void ResurrectStateCharacter::init(InputInfo aInputInfo) {
    aPriestCell = nullptr;
    finalized = false;
    movement.reset();
}
```

jul 21, 20 15:20 **PursuitStateCreature.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_PURSUITSTATECREATURE_H
#define ARGENTUM_TALLER_PURSUITSTATECREATURE_H


#include <MovementCreature.h>
#include "State.h"
#include "../Creature.h"
#include "StateCreature.h"

class PursuitStateCreature : public StateCreature {
private:
    uint pursuitId;
    MovementCreature movement;
    bool canAttack = false;

public:
    explicit PursuitStateCreature();

    ~PursuitStateCreature() override;

    bool isOnPursuit(uint pursuitId) override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    void init(InputInfo aInputInfo) override;

    bool isAttacking() override;
};


#endif //ARGENTUM_TALLER_PURSUITSTATECREATURE_H
```

jul 21, 20 15:20 **PursuitStateCreature.cpp** Page 1/2

```cpp
#include "PursuitStateCreature.h"
#include "StillStateCreature.h"
#include "../GameCharacter.h"

PursuitStateCreature::~PursuitStateCreature() = default;

PursuitStateCreature::PursuitStateCreature() : StateCreature(), pursuitId(0) {
    stateId = StateID::Pursuit;
    finalized = false;
}

void PursuitStateCreature::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {

    std::shared_ptr<Creature> aCreature = std::dynamic_pointer_cast<Creature>(gameObjects.at(id));
    try {
        std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(pursuitId));
        std::shared_ptr<Cell> creatureCell = aCreature→getActualCell();
        std::shared_ptr<Cell> enemyCell = aCharacter→getActualCell();
        if (¬movement.hasStart()) {
            if (enemyCell→getNestId() ≡ creatureCell→getNestId()) {
                if (board.getDistance(creatureCell, enemyCell) ≡ 1) {
                    canAttack = true;
                    finalized = true;
                    aCreature→setDirection(board.getDirection(creatureCell, enemyCell));
                } else {
                    std::shared_ptr<Cell> newCell;
                    newCell = board.getBestCell(creatureCell, enemyCell, false);
                    if (newCell ≠ aCreature→getActualCell()) {
                        Direction aDirection = board.getDirection(creatureCell, newCell);
                        aCreature→setDirection(aDirection);
                        movement.start(board.getPointFromCell(aCreature→getActualCell()), aDirection, aCreature→getCreatureId());
                        aCreature→getActualCell()→free();
                        newCell→occupied(id);
                        aCreature→setCell(newCell);
                    }
                }
            } else {
                finalized = true;
            }
        } else {
            aCreature→setPoint(movement.doStep());
            if (movement.isOver()) {
                movement.reset();
            }
        }
    } catch (const std::out_of_range& e) {
        canAttack = false;
        finalized = true;
    }

}

bool PursuitStateCreature::isOnPursuit(uint aPursuitId) {
    return pursuitId ≡ aPursuitId;
}

bool PursuitStateCreature::isAttacking() {
    return false;
}

```

jul 21, 20 15:20 **PursuitStateCreature.cpp** Page 2/2

```cpp
StateID PursuitStateCreature::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (canAttack) {
        nextStateId = StateID::Attack;
    }
    return nextStateId;
}

StateID PursuitStateCreature::getResetStateID() {
    return StateID::Pursuit;
}

void PursuitStateCreature::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
    pursuitId = aInputInfo.aditional;
    movement.reset();
    canAttack = false;
    finalized = false;
}
```

jul 21, 20 15:20 **MoveStateCreature.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_MOVESTATECREATURE_H
#define ARGENTUM_TALLER_MOVESTATECREATURE_H


#include <MovementCreature.h>
#include "State.h"
#include "StateCreature.h"

class MoveStateCreature : public StateCreature {
private:
    Direction direction;
    MovementCreature movement;
public:
    MoveStateCreature();
    ~MoveStateCreature() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    bool isOnPursuit(uint pursuitId) override;

    void init(InputInfo aInputInfo) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    bool isAttacking() override;
};


#endif //ARGENTUM_TALLER_MOVESTATECREATURE_H
```

jul 21, 20 15:20 **MoveStateCreature.cpp** Page 1/2

```cpp
#include <iostream>
#include "MoveStateCreature.h"
#include "StillStateCreature.h"
#include "../Creature.h"

MoveStateCreature::~MoveStateCreature() = default;

MoveStateCreature::MoveStateCreature() : StateCreature(){
    stateId = StateID::Move;
    direction = Direction::down;
    finalized = false;
}

void MoveStateCreature::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {
    std::shared_ptr<Creature> aCreature = std::dynamic_pointer_cast<Creature>(gameObjects.at(id));
    if (¬movement.hasStart()) {
        std::shared_ptr<Cell> newCell;
        aCreature→setDirection(direction);
        if (board.creatureCanMove(aCreature→getActualCell(), direction)) {
            newCell = board.getNextCell(aCreature→getActualCell(), direction);
            movement.start(board.getPointFromCell(aCreature→getActualCell()), direction, aCreature→getCreatureId());
            aCreature→getActualCell()→free();
            newCell→occupied(id);
            aCreature→setCell(newCell);
        } else {
            finalized = true;
        }
    } else {
        aCreature→setPoint(movement.doStep());
        if (movement.isOver()) {
            finalized = true;
        }
    }
}

bool MoveStateCreature::isOnPursuit(uint pursuitId) {
    return false;
}

bool MoveStateCreature::isAttacking() {
    return false;
}

StateID MoveStateCreature::getNextStateID(InputInfo info) {
    return StateID::Move;
}

StateID MoveStateCreature::getResetStateID() {
    return StateID::Move;
}

void MoveStateCreature::init(InputInfo aInputInfo) {
    movement.reset();
    inputInfo = aInputInfo;
    switch(aInputInfo.input) {
        case InputID::up:
            direction = Direction::up;
            break;
        case InputID::down:
            direction = Direction::down;
            break;
        case InputID::left:
            direction = Direction::left;
```

jul 21, 20 15:20 **MoveStateCreature.cpp** Page 2/2

```
64              break;
65          case InputID::right:
66              direction = Direction::right;
67              break;
68          default:
69              direction = Direction::up;
70              break;
71      }
72      finalized = false;
73  }
```

jul 21, 20 15:20 **MoveStateCharacter.h** Page 1/1

```
1   #ifndef ARGENTUM_TALLER_MOVESTATECHARACTER_H
2   #define ARGENTUM_TALLER_MOVESTATECHARACTER_H
3
4
5   #include <MovementCharacter.h>
6   #include "State.h"
7   #include "StateCharacter.h"
8
9   class MoveStateCharacter : public StateCharacter {
10  private:
11      Direction direction;
12      MovementCharacter movement;
13
14  public:
15      MoveStateCharacter();
16
17      ~MoveStateCharacter() override;
18
19      void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObjec
    t>> &gameObjects, Board &board) override;
20
21      bool isAttacking() override;
22
23      bool isMeditating() override;
24
25      void init(InputInfo aInputInfo) override;
26
27      StateID getNextStateID(InputInfo info) override;
28
29      StateID getResetStateID() override;
30  };
31
32
33  #endif //ARGENTUM_TALLER_MOVESTATECHARACTER_H
```

jul 21, 20 15:20 **MoveStateCharacter.cpp** Page 1/2

```cpp
#include <iostream>
#include "MoveStateCharacter.h"
#include "../GameCharacter.h"
#include "StillStateCharacter.h"
#include "../Creature.h"


MoveStateCharacter::~MoveStateCharacter() = default;

MoveStateCharacter::MoveStateCharacter() : StateCharacter() {
    stateId = StateID::Move;
    direction = Direction::down;
}

void MoveStateCharacter::performTask(uint id,
                                     std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects,
                                     Board &board) {

    std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(id));
    if (¬movement.hasStart()) {
        aCharacter→setDirection(direction);
        std::shared_ptr<Cell> newCell;
        if (board.characterCanMove(aCharacter→getActualCell(), direction)) {
            newCell = board.getNextCell(aCharacter→getActualCell(), direction);
            movement.start(board.getPointFromCell(aCharacter→getActualCell()), direction, aCharacter→getRace());
            aCharacter→getActualCell()→free();
            newCell→occupied(id);
            aCharacter→setCell(newCell);
            uint nestId = newCell→getNestId();
            if (nestId ≠ 0 ∧ ¬aCharacter→isDead()) {
                std::shared_ptr<Creature> aCreature;
                for(auto &creatureId : board.getCreaturesInNest(nestId)) {
                    aCreature = std::dynamic_pointer_cast<Creature>(gameObjects.at(creatureId));
                    aCreature→notify(id);
                }
            }
        } else {
            finalized = true;
        }
    } else {
        aCharacter→setPoint(movement.doStep());
        if (movement.isOver()) {
            finalized = true;
        }
    }

}

StateID MoveStateCharacter::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextStateId = StateID::Move;
    } else if(info.input ≡ InputID::stopMove) {
        nextStateId = StateID::Still;
    }
    return nextStateId;
}

StateID MoveStateCharacter::getResetStateID() {
    return StateID::Move;
}
```

jul 21, 20 15:20 **MoveStateCharacter.cpp** Page 2/2

```cpp

bool MoveStateCharacter::isAttacking() {
    return false;
}

bool MoveStateCharacter::isMeditating() {
    return false;
}

void MoveStateCharacter::init(InputInfo aInputInfo) {
    movement.reset();
    switch(aInputInfo.input) {
        case InputID::up:
            direction = Direction::up;
            break;
        case InputID::down:
            direction = Direction::down;
            break;
        case InputID::left:
            direction = Direction::left;
            break;
        case InputID::right:
            direction = Direction::right;
            break;
        default:
            direction = Direction::down;
            break;
    }
    inputInfo = aInputInfo;
    finalized = false;
}
```

jul 21, 20 15:20 **MeditateStateCharacter.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_MEDITATESTATECHARACTER_H
#define ARGENTUM_TALLER_MEDITATESTATECHARACTER_H


#include "StateCharacter.h"

class MeditateStateCharacter : public StateCharacter {
public:
    MeditateStateCharacter();

    ~MeditateStateCharacter() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    bool isAttacking() override;

    bool isMeditating() override;

    void init(InputInfo aInputInfo) override;

};


#endif //ARGENTUM_TALLER_MEDITATESTATECHARACTER_H
```

jul 21, 20 15:20 **MeditateStateCharacter.cpp** Page 1/1

```
#include "MeditateStateCharacter.h"

MeditateStateCharacter::~MeditateStateCharacter() = default;

MeditateStateCharacter::MeditateStateCharacter() {
    finalized = true;
    stateId = StateID::Meditate;
}

void MeditateStateCharacter::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects,
                                         Board &board) {
}

bool MeditateStateCharacter::isAttacking() {
    return false;
}

bool MeditateStateCharacter::isMeditating() {
    return true;
}

StateID MeditateStateCharacter::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextStateId = StateID::Move;
    }
    return nextStateId;
}

StateID MeditateStateCharacter::getResetStateID() {
    return StateID::Meditate;
}

void MeditateStateCharacter::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
}
```

jul 21, 20 15:20 **InteractStateCharacter.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_INTERACTSTATECHARACTER_H
#define ARGENTUM_TALLER_INTERACTSTATECHARACTER_H

#include "StateCharacter.h"

class InteractStateCharacter: public StateCharacter {
private:
    bool interacting = false;
    std::shared_ptr<GameObject> aNpc = nullptr;
public:
    InteractStateCharacter();

    ~InteractStateCharacter() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    void init(InputInfo aInputInfo) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    bool isMeditating() override;

    bool isAttacking() override;

};

#endif
```

jul 21, 20 15:20 **InteractStateCharacter.cpp** Page 1/2

```cpp
#include "InteractStateCharacter.h"
#include "../GameCharacter.h"

InteractStateCharacter::InteractStateCharacter() : StateCharacter() {
    stateId = StateID::Interact;
}

InteractStateCharacter::~InteractStateCharacter() = default;

void InteractStateCharacter::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {

    std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(id));
    std::shared_ptr<Cell> npcCell = board.getCellFromPoint(inputInfo.position);
    if (¬interacting) {
        if (npcCell→getGameObjectId() ≠ 0 ∧ npcCell ≠ aCharacter→getActualCell()) {
            aNpc = gameObjects.at(npcCell→getGameObjectId());
            if (board.getDistance(npcCell, aCharacter→getActualCell()) ≡ 1) {
                NPCInfo info = aNpc→interact(*aCharacter,inputInfo);
                if (info.type ≠ 0) {
                    aCharacter→setInteractInfo(info);
                    interacting = true;
                } else {
                    finalized = true;
                }
            }
        } else {
            finalized = true;
        }
    } else {
        if (aCharacter→hasAnInputInfo()) {
            InputInfo info = aCharacter→getNextInputInfo();
            if (aCharacter→isDead() ∧ ¬(info.input ≡ InputID::resurrect)) {
                finalized = true;
                return;
            }
            if (info.input ≡ InputID::buy ∨ info.input ≡ InputID::cure ∨
                info.input ≡ InputID::sell ∨ info.input ≡ InputID::resurrect ∨
                info.input ≡ InputID::retireItem ∨ info.input ≡ InputID::retireGold ∨
                info.input ≡ InputID::depositItem ∨ info.input ≡ InputID::depositGold) {
                try {
                    aCharacter→setInteractInfo(aNpc→interact(*aCharacter, info));
                } catch (Exception &e) {
                    finalized = true;
                }
            } else {
                finalized = true;
            }
        }
    }
}

StateID InteractStateCharacter::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextStateId = StateID::Move;
    }
    return nextStateId;
}

```

jul 21, 20 15:20 **InteractStateCharacter.cpp** Page 2/2

```
61 StateID InteractStateCharacter::getResetStateID() {
62     return StateID::Still;
63 }
64
65 bool InteractStateCharacter::isAttacking() {
66     return false;
67 }
68
69 bool InteractStateCharacter::isMeditating() {
70     return false;
71 }
72
73 void InteractStateCharacter::init(InputInfo aInputInfo) {
74     interacting = false;
75     aNpc = nullptr;
76     inputInfo = aInputInfo;
77     finalized = false;
78 }
79
80
```

jul 21, 20 15:20 **EquipStateCharacter.h** Page 1/1

```
1  #ifndef ARGENTUM_TALLER_EQUIPSTATECHARACTER_H
2  #define ARGENTUM_TALLER_EQUIPSTATECHARACTER_H
3
4
5  #include "StateCharacter.h"
6
7  class EquipStateCharacter : public StateCharacter {
8  private:
9      int itemToChange{0};
10 public:
11     explicit EquipStateCharacter();
12
13     ~EquipStateCharacter() override;
14
15     void
16     performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &
   gameObjects, Board &board) override;
17
18     StateID getNextStateID(InputInfo info) override;
19
20     StateID getResetStateID() override;
21
22     bool isMeditating() override;
23
24     void init(InputInfo aInputInfo) override;
25
26     bool isAttacking() override;
27 };
28
29
30 #endif //ARGENTUM_TALLER_EQUIPSTATECHARACTER_H
```

jul 21, 20 15:20 **EquipStateCharacter.cpp** Page 1/1

```cpp
#include "EquipStateCharacter.h"
#include "../GameCharacter.h"

EquipStateCharacter::~EquipStateCharacter() = default;

EquipStateCharacter::EquipStateCharacter() : StateCharacter() {
    finalized = true;
    stateId = StateID::Equip;
}

void EquipStateCharacter::performTask(uint id,
        std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {

    std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(id));
    if (inputInfo.input ≡ InputID::unequipItem) {
        aCharacter→unequipItem(inputInfo.aditional);
    } else {
        if (inputInfo.aditional ≠ 0) {
            aCharacter→equipItem(inputInfo.aditional);
        }
    }
}

bool EquipStateCharacter::isAttacking() {
    return false;
}

bool EquipStateCharacter::isMeditating() {
    return false;
}

StateID EquipStateCharacter::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextStateId = StateID::Move;
    }
    return nextStateId;
}

StateID EquipStateCharacter::getResetStateID() {
    return StateID::Still;
}

void EquipStateCharacter::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
}
```

jul 21, 20 15:20 **AttackStateCreature.h** Page 1/1

```cpp
#ifndef ARGENTUM_TALLER_ATTACKSTATECREATURE_H
#define ARGENTUM_TALLER_ATTACKSTATECREATURE_H


#include "StateCreature.h"

class AttackStateCreature : public StateCreature {
private:
    uint enemyId = 0;
    bool enemyIsDead;
    uint8_t timeBetweenAttacks = 0;
    std::shared_ptr<GameObject> aEnemy = nullptr;
public:
    AttackStateCreature();

    ~AttackStateCreature() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    bool isOnPursuit(uint pursuitId) override;

    void init(InputInfo aInputInfo) override;

    bool isAttacking() override;
};


#endif //ARGENTUM_TALLER_ATTACKSTATECREATURE_H
```

jul 21, 20 15:20 **AttackStateCreature.cpp** Page 1/2

```
#include "AttackStateCreature.h"
#include "../Creature.h"

AttackStateCreature::~AttackStateCreature() = default;

AttackStateCreature::AttackStateCreature() : StateCreature(), enemyIsDead(false)
 {
    stateId = StateID::Attack;
    finalized = false;
}

void AttackStateCreature::performTask(uint id, std::unordered_map<uint, std::sha
red_ptr<GameObject>> &gameObjects, Board &board) {

    if (timeBetweenAttacks ≡ 0) {
        timeBetweenAttacks = 30;
        std::shared_ptr<Creature> aCreature = std::dynamic_pointer_cast<Creature
>(gameObjects.at(id));
        try {
            aEnemy = gameObjects.at(enemyId);
            if (aEnemy→isDead()) {
                enemyIsDead = true;
                finalized = true;
                return;
            }
        } catch (const std::exception& e) {
            enemyIsDead = true;
            finalized = true;
            return;
        }
        std::shared_ptr<Cell> creatureCell = aCreature→getActualCell();
        std::shared_ptr<Cell> enemyCell = aEnemy→getActualCell();

        if (board.getDistance(creatureCell, enemyCell) ≡ GameStatsConfig::getWea
ponDistance(WeaponID::Nothing)) {
            aEnemy→receiveDamage(GameStatsConfig::getDamage(aCreature→getCreat
ureId()),
                    WeaponID::Nothing);
        } else {
            if (aEnemy ≠ nullptr) {
                aEnemy→setInteractWeapon(WeaponID::Nothing);
            }
            finalized = true;
        }
    } else {
        timeBetweenAttacks−−;
    }
}

bool AttackStateCreature::isOnPursuit(uint pursuitId) {
    return false;
}

bool AttackStateCreature::isAttacking() {
    return true;
}

StateID AttackStateCreature::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (¬enemyIsDead) {
        nextStateId = StateID::Pursuit;
    }
    return nextStateId;
}

StateID AttackStateCreature::getResetStateID() {
```

jul 21, 20 15:20 **AttackStateCreature.cpp** Page 2/2

```
    return StateID::Still;
}

void AttackStateCreature::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
    enemyId = aInputInfo.aditional;
    enemyIsDead = false;
    timeBetweenAttacks = 0;
    aEnemy = nullptr;
    finalized = false;
}
```

jul 21, 20 15:20 **AttackStateCharacter.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_ATTACKSTATECHARACTER_H
#define ARGENTUM_TALLER_ATTACKSTATECHARACTER_H


#include "State.h"
#include "StateCharacter.h"

class AttackStateCharacter : public StateCharacter {
private:
    uint8_t timeBetweenAttacks = 0;
    bool enemyReceiveDamage = false;
    std::shared_ptr<GameObject> aEnemy = nullptr;
public:
    AttackStateCharacter();

    ~AttackStateCharacter() override;

    void performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    StateID getNextStateID(InputInfo info) override;

    StateID getResetStateID() override;

    void init(InputInfo aInputInfo) override;

    bool isAttacking() override;

    bool isMeditating() override;
};


#endif //ARGENTUM_TALLER_ATTACKSTATECHARACTER_H
```

jul 21, 20 15:20 **AttackStateCharacter.cpp** Page 1/2

```cpp
#include "AttackStateCharacter.h"
#include "../GameCharacter.h"

AttackStateCharacter::~AttackStateCharacter() = default;

AttackStateCharacter::AttackStateCharacter() : StateCharacter() {
    stateId = StateID::Attack;
}

void AttackStateCharacter::performTask(uint id, std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects,
                                       Board &board) {

    std::shared_ptr<GameCharacter> aCharacter = std::dynamic_pointer_cast<GameCharacter>(gameObjects.at(id));
    if (timeBetweenAttacks ≡ 0) {
        timeBetweenAttacks = 20;
        std::shared_ptr<Cell> enemyCell = board.getCellFromPoint(inputInfo.position);
        if (enemyCell ≡ aCharacter→getActualCell() ∧ aCharacter→getWeapon() ≡ WeaponID::ElficFlaute) {
            aCharacter→restoreHealth();
            aEnemy = aCharacter;
            return;
        }
        if (enemyCell→getGameObjectId() ≠ 0 ∧ enemyCell ≠ aCharacter→getActualCell()) {
            try {
                aEnemy = gameObjects.at(enemyCell→getGameObjectId());
                if (¬aEnemy→isDead() ∧
                    board.getDistance(aCharacter→getActualCell(), enemyCell) ≤ GameStatsConfig::getWeaponDistance(aCharacter→getWeapon()) ∧
                    aCharacter→canUseWeapon() ∧ aEnemy→canBeAttacked(aCharacter→getLevel())) {

                    float damage = GameStatsConfig::getDamage(aCharacter→getRace(), aCharacter→getWeapon());
                    aCharacter→consumeMana();
                    aEnemy→receiveDamage(damage, aCharacter→getWeapon());
                    aCharacter→gainExp(aEnemy→isDead() ?
                        GameStatsConfig::getAdditionalExp(damage, aEnemy→getMaxLife(), aCharacter→getLevel(), aEnemy→getLevel()) :
                        GameStatsConfig::getExp(damage, aCharacter→getLevel(), aEnemy→getLevel()));
                    if (aCharacter→getExp() ≥ GameStatsConfig::getNextLevelLimit(aCharacter→getLevel())) {
                        aCharacter→upLevel();
                    }
                    enemyReceiveDamage = true;
                }
            } catch (const std::exception& e) {
                std::cout << "Cannot get Enemy" << std::endl;
            }
        }
        if (¬enemyReceiveDamage) {
            finalized = true;
        }
    } else {
        timeBetweenAttacks−−;
        if (timeBetweenAttacks ≡ 0) {
            finalized = true;
            if (aEnemy ≠ nullptr) {
                aEnemy→setInteractWeapon(WeaponID::Nothing);
            }
        }
    }
```

jul 21, 20 15:20 **AttackStateCharacter.cpp** Page 2/2

```
}

bool AttackStateCharacter::isAttacking() {
    return true;
}

bool AttackStateCharacter::isMeditating() {
    return false;
}

StateID AttackStateCharacter::getNextStateID(InputInfo info) {
    StateID nextStateId = StateID::Still;
    if (info.input ≡ InputID::up ∨ info.input ≡ InputID::down ∨
        info.input ≡ InputID::left ∨ info.input ≡ InputID::right) {
        nextStateId = StateID::Move;
    }
    return nextStateId;
}

StateID AttackStateCharacter::getResetStateID() {
    return StateID::Still;
}

void AttackStateCharacter::init(InputInfo aInputInfo) {
    inputInfo = aInputInfo;
    finalized = false;
    aEnemy = nullptr;
    timeBetweenAttacks = 0;
    enemyReceiveDamage = false;
}
```

jul 21, 20 15:20 **server_main.cpp** Page 1/1

```
#include <iostream>
#include "World.h"
#include "../common/JsonReader.h"
#include "PlayerAcceptor.h"
#include <string>

#define EXIT "q"
#define INVALIDARGUMENTS "Error argumentos invÃ¡lidos: argentum_server <path configuracion>"

int main(int argc, char* argv[]) {
    if (argc ≠ 2) {
        std::cout << INVALIDARGUMENTS <<std::endl;
        return 1;
    }
    rapidjson::Document jsonConfigStats = JsonReader::read(argv[1]);
    GameStatsConfig gameStatsConfig(jsonConfigStats);

    World world(gameStatsConfig);
    PlayerAcceptor acceptor(gameStatsConfig.getPort(),world);

    acceptor.start();
    world.start();

    std::string input;
  while (input ≠ EXIT) {
    std::cin >> input;
  }

    world.stop();
  world.join();
    std::cout << "Server Cerrado" << std::endl;
    acceptor.stop();
    acceptor.join();
    return 0;
}

```

jul 21, 20 15:20 **ServerItem.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_SERVERITEM_H
#define ARGENTUM_TALLER_SERVERITEM_H


#include "../common/Identificators.h"

class ServerItem {
private:
    ItemsInventoryID id;
    int16_t amount;
public:
    ServerItem(ItemsInventoryID id, int16_t amount);

    ItemsInventoryID getId() const;

    friend bool operator≡(ServerItem &thisItem, const ServerItem& otherItem);

    friend bool operator≠(ServerItem &thisItem, const ServerItem& otherItem);

    void increaseAmount();

    void decreseAmount();

    virtual ~ServerItem();

};


#endif //ARGENTUM_TALLER_SERVERITEM_H
```

jul 21, 20 15:20 **ServerItem.cpp** Page 1/1

```cpp
#include "ServerItem.h"

ServerItem::~ServerItem() = default;

ServerItem::ServerItem(ItemsInventoryID id, int16_t amount) : id(id), amount(amo
unt) {}

ItemsInventoryID ServerItem::getId() const {
    return id;
}

void ServerItem::increaseAmount() {
    amount++;
}

void ServerItem::decreseAmount() {
    amount−−;
    if (amount ≡ 0) {
        id = ItemsInventoryID::Nothing;
        amount++;
    }
}

bool operator≡(ServerItem &thisItem, const ServerItem &otherItem) {
    return thisItem.getId() ≡ otherItem.getId();
}

bool operator≠(ServerItem &thisItem, const ServerItem &otherItem) {
    return ¬(thisItem ≡ otherItem);
}
```

jul 21, 20 15:20 **Profession.h** Page 1/1

```
#ifndef PROFESSION_H
#define PROFESSION_H

#include <vector>
#include "../common/Identificators.h"
#include "GameObject.h"
#include "GameCharacter.h"

class Profession {
protected:
    std::vector<ActionsProfessionID> actions;
public:
    Profession() : actions() {}

    virtual ~Profession(){}

    std::vector<ActionsProfessionID> getActions() const {
        return actions;
    }

    virtual void processInput(GameCharacter& character, InputInfo inputInfo) = 0
;

    virtual NPCInfo getInfo(uint id) = 0;
};

#endif
```

jul 21, 20 15:20 **Priest.h** Page 1/1

```
#ifndef PRIEST_H
#define PRIEST_H

#include "Profession.h"
#include <unordered_map>
#include <vector>
#include "../common/Identificators.h"
#include "states/State.h"

class Priest : public Profession {
private:
    Priest();
    std::unordered_map<ItemsInventoryID, uint,std::hash<ItemsInventoryID>> items
;

    static Priest* priest;

    ItemsInventoryID buyItem(ItemsInventoryID idItem, uint* balance) const;

    std::unordered_map<ItemsInventoryID,uint> getItems() const;
public:
    static Priest* getInstance();

    void init(const std::map<ItemsInventoryID, ItemInfo>& itemsToInit);

    virtual NPCInfo getInfo(uint id);

    void processInput(GameCharacter &character, InputInfo inputInfo) override;

    virtual ~Priest();
};

#endif
```

jul 21, 20 15:20 **Priest.cpp** Page 1/2

```
#include "Priest.h"

Priest* Priest::priest = nullptr;

Priest::Priest() :items() {
    this→actions.push_back(ActionsProfessionID::Cure);
    this→actions.push_back(ActionsProfessionID::Resurrect);
    this→actions.push_back(ActionsProfessionID::Buy);
}


Priest* Priest::getInstance() {
    if (priest ≡ nullptr)
        priest = new Priest();
    return priest;
}

void Priest::init(const std::map<ItemsInventoryID, ItemInfo>&itemsToInit) {
    for (auto &iter: itemsToInit){
        if(iter.second.type ≡ "Potion")
            this→items.insert({iter.first, iter.second.goldCost});
    }
}


ItemsInventoryID Priest::buyItem(ItemsInventoryID idItem, uint* balance) const {
    auto iter = items.find(idItem);
    if (*balance < (*iter).second)
        return ItemsInventoryID::Nothing;
    *balance −= (*iter).second;
    return idItem;
}

std::unordered_map<ItemsInventoryID,uint> Priest::getItems() const {
    return items;
}

NPCInfo Priest::getInfo(uint id) {
    NPCInfo info;
    info.type = 2;
    info.actions = actions;
    info.gold = 0;
    info.items = getItems();
    return info;
}

void Priest::processInput(GameCharacter &character, InputInfo inputInfo) {
    uint goldAmount = 0;
        switch (inputInfo.input) {
        case InputID::buy:
            goldAmount = character.getGoldAmount();
            if (¬character.inventoryIsFull()) {
                ItemsInventoryID aItem = buyItem(ItemsInventoryID(inputInfo.adit
ional), &goldAmount);
                if (aItem ≠ ItemsInventoryID::Nothing) {
                    character.addItemToInventory(aItem);
                    character.setGoldAmount(goldAmount);
                }
            }
            break;
        case InputID::resurrect:
            if (character.isDead())
                character.cure();
            break;
        case InputID::cure:
            character.cure();
```

jul 21, 20 15:20 **Priest.cpp** Page 2/2

```
            break;
        default:
            break;
        }
}

Priest::~Priest() {}
```

jul 21, 20 15:20 **PlayerAcceptor.h** Page 1/1

```h
#ifndef PLAYERACCEPTOR_H
#define PLAYERACCEPTOR_H

#include "../common/Thread.h"
#include "../common/Socket.h"
#include "ThPlayer.h"
#include "World.h"
#include "ThLobbyPlayer.h"

class PlayerAcceptor: public Thread {
private:
    Socket socket;
    World& world;
    std::vector<ThLobbyPlayer*> players;
    std::atomic<bool> keepTalking;

    void clear_finished_games();
    void stop_players();
public:
    explicit PlayerAcceptor(const std::string& port, World& world);

    virtual void run();

    void stop();

    virtual ~PlayerAcceptor();
};

#endif
```

jul 21, 20 15:20 **PlayerAcceptor.cpp** Page 1/1

```cpp
#include "PlayerAcceptor.h"
#include <sys/socket.h>
#include <iostream>
#define MAX_WAITING 20

PlayerAcceptor::PlayerAcceptor(const std::string& port, World& world) : socket()
,
    world(world), players(), keepTalking(true){
        this→socket.bind_and_listen(port.data(), MAX_WAITING);
    std::cout << "Se hizo el bind correctamente " << port.data() << std::endl;
}

void PlayerAcceptor::clear_finished_games() {
  std::vector<ThLobbyPlayer*>::iterator iter;
    iter = this→players.begin();
    while (iter ≠ this→players.end()){
        if (¬(*iter)→is_alive()){
            std::cout << "Se cerro el lobby de un jugador" << std::endl;
            (*iter)→join();
            delete (*iter);
            iter = this→players.erase(iter);
        } else {
            iter++;
        }
    }
}

void PlayerAcceptor::run() {
    while (this→keepTalking) {
        try{
            Socket socketPlayer = this→socket.accept();
            auto player = new ThLobbyPlayer(std::move(socketPlayer), world);
            player→start();
            this→players.push_back(player);
            clear_finished_games();
        } catch (...) {

        }
    }
}

void PlayerAcceptor::stop_players() {
    for (auto & player : this→players) {
    player→stop();
        player→join();
        delete player;
  }
}

void PlayerAcceptor::stop() {
    this→keepTalking = false;
  this→socket.shutdown(SHUT_RDWR);
  this→socket.close();
  stop_players();
}

PlayerAcceptor::~PlayerAcceptor() = default;
```

jul 21, 20 15:20 **ObjectItem.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_OBJECTITEM_H
#define ARGENTUM_TALLER_OBJECTITEM_H


#include "GameObject.h"

class ObjectItem : public GameObject {
private:
    int timeToBeRemove = 45 * 300;
    int amount;
    ItemsInventoryID itemId;
public:
    ObjectItem(uint id, const Point &initialPoint, const std::shared_ptr<Cell> &initialCell, const DropItem& dropItem);

    float getMaxLife() override;

    void update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    CharacterStateID getStateId() override;

    bool isDead() override;

    PlayerInfo getPlayerInfo() override;

    bool hasAnInputInfo() override;

    InputInfo getNextInputInfo() override;

    bool isItem() override;

    void take();

    int getAmount() const;

    ItemsInventoryID getItemId() const;

    bool canDropsItems() override;

    std::vector<DropItem> getDrop() override;

    void receiveDamage(float damage, WeaponID weaponId) override;

    NPCInfo interact(GameObject &character, InputInfo input) override;

    bool isReadyToRemove() override;

    void remove(Board &board) override;

    virtual ~ObjectItem();

    bool canBeAttacked(int enemyLevel) const;
};


#endif //ARGENTUM_TALLER_OBJECTITEM_H
```

jul 21, 20 15:20 **ObjectItem.cpp** Page 1/2

```cpp
#include "ObjectItem.h"

ObjectItem::ObjectItem(uint id, const Point &initialPoint, const std::shared_ptr<Cell> &initialCell, const DropItem& dropItem)
: GameObject(id, initialPoint, initialCell), amount(dropItem.getAmount()), itemId(dropItem.getId()) {
    std::string stringItemID = std::to_string((int)itemId);
    textureHashId = "ht00|h00|b00|s00|w00|i" + (stringItemID.size() == 2 ? stringItemID : "0" + stringItemID);
}

float ObjectItem::getMaxLife() {
    return 0;
}

void ObjectItem::update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {
    timeToBeRemove--;
}

CharacterStateID ObjectItem::getStateId() {
    return CharacterStateID::Still;
}

bool ObjectItem::isDead() {
    return false;
}

void ObjectItem::receiveDamage(float damage, WeaponID weaponId) {}

NPCInfo ObjectItem::interact(GameObject &character, InputInfo input) {
    return NPCInfo();
}

bool ObjectItem::isReadyToRemove() {
    return timeToBeRemove <= 0;
}

void ObjectItem::remove(Board &board) {
    getActualCell()->removeItem();
}

std::vector<DropItem> ObjectItem::getDrop() {
    return std::vector<DropItem>();
}

bool ObjectItem::isItem() {
    return true;
}

bool ObjectItem::canDropsItems() {
    return false;
}

ObjectItem::~ObjectItem() = default;

int ObjectItem::getAmount() const {
    return amount;
}

ItemsInventoryID ObjectItem::getItemId() const {
    return itemId;
}

void ObjectItem::take() {
    timeToBeRemove = 0;
```

jul 21, 20 15:20 **ObjectItem.cpp** Page 2/2

```
}

bool ObjectItem::canBeAttacked(int enemyLevel) const {
    return false;
}

PlayerInfo ObjectItem::getPlayerInfo() {
    return GameObject::getPlayerInfo();
}

bool ObjectItem::hasAnInputInfo() {
    return false;
}

InputInfo ObjectItem::getNextInputInfo() {
    InputInfo info;
    info.input = InputID::nothing;
    return info;
}
```

jul 21, 20 15:20 **NPCServer.h** Page 1/1

```
#ifndef NPC_H
#define NPC_H

#include "GameObject.h"
#include "GameCharacter.h"
#include "Profession.h"
#include "states/State.h"

class NPCServer : public GameObject{
private:
    Profession* profession;
public:
  NPCServer(uint id, const std::string& type, Point initialPoint, std::shared_ptr<Cell> initialCell);

    virtual ~NPCServer();

    void update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override ;

    CharacterStateID getStateId() override;

    NPCInfo interact(GameObject& character, InputInfo input) override;

    void receiveDamage(float damage, WeaponID weaponId) override;

    bool isItem() override;

    bool canDropsItems() override;

    PlayerInfo getPlayerInfo() override;

    bool hasAnInputInfo() override;

    InputInfo getNextInputInfo() override;

    float getMaxLife() override;

    std::vector<DropItem> getDrop() override;

    bool isDead() override;

    void remove(Board &board) override;

    bool isReadyToRemove() override;

    bool canBeAttacked(int enemyLevel) const override ;
};

#endif
```

jul 21, 20 15:20 **NPCServer.cpp** Page 1/2

```cpp
#include "NPCServer.h"

#include <utility>
#include "Banker.h"
#include "Merchant.h"
#include "Priest.h"
#include "states/StillStateCharacter.h"

NPCServer::~NPCServer() = default;

NPCServer::NPCServer(uint id, const std::string& type, Point initialPoint, std::shared_ptr<Cell> initialCell) :
    GameObject(id, initialPoint, std::move(initialCell)) {
    if (type == "banker") {
        textureHashId = "ht00|h00|b08|s00|w00";
        this->profession = Banker::getInstance();
    } else if (type == "merchant") {
        textureHashId = "ht00|h00|b09|s00|w00";
        this->profession = Merchant::getInstance();
    } else {
        cell->addPriest();
        textureHashId = "ht00|h05|b10|s00|w00";
        this->profession = Priest::getInstance();
    }
}

void NPCServer::update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {}

NPCInfo NPCServer::interact(GameObject& character, InputInfo inputInfo) {
    if (inputInfo.input != InputID::selectTarget) {
        this->profession->processInput(dynamic_cast<GameCharacter &>(character), inputInfo);
    }
    NPCInfo info = this->profession->getInfo(character.getId());
    return info;
}

CharacterStateID NPCServer::getStateId() {
    return CharacterStateID::Still;
}

bool NPCServer::isReadyToRemove() {
    return false;
}

bool NPCServer::isDead() {
    return false;
}

void NPCServer::remove(Board &board) {
    cell->free();
}

float NPCServer::getMaxLife() {
    return 0;
}

void NPCServer::receiveDamage(float damage, WeaponID weaponId) {
}

std::vector<DropItem> NPCServer::getDrop() {
    return std::vector<DropItem>();
}

bool NPCServer::isItem() {
```

jul 21, 20 15:20 **NPCServer.cpp** Page 2/2

```cpp
    return false;
}

bool NPCServer::canDropsItems() {
    return false;
}

bool NPCServer::canBeAttacked(int enemyLevel) const {
    return false;
}

PlayerInfo NPCServer::getPlayerInfo() {
    return GameObject::getPlayerInfo();
}

bool NPCServer::hasAnInputInfo() {
    return false;
}

InputInfo NPCServer::getNextInputInfo() {
    InputInfo info;
    info.input = InputID::nothing;
    return info;
}
```

jul 21, 20 15:20 **Node.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_NODE_H
#define ARGENTUM_TALLER_NODE_H


#include <memory>
#include "Cell.h"

class Node {
private:
    int parent;
    int g;
    int h;
    int id;
    std::shared_ptr<Cell> cell;
public:
    Node(int parent, int g, int h, std::shared_ptr<Cell> cell);

    const std::shared_ptr<Cell> &getCell() const;

    friend bool operator≡(const Node& node1, const Node& node2);

    friend bool operator≠(const Node& node1, const Node& node2);

    int getF();

    void setParent(int parent);

    int getParent() const;

    int getId() const;

    int getG() const;

    void setG(int g);

    int getH() const;

    void setH(int h);

    virtual ~Node();

};


#endif //ARGENTUM_TALLER_NODE_H
```

jul 21, 20 15:20 **Node.cpp** Page 1/1

```cpp
#include "Node.h"

#include <utility>

Node::Node(int parent, int g, int h, std::shared_ptr<Cell> aCell) :
parent(parent), g(g), h(h), cell(std::move(aCell)) {
    id = std::stoi(std::to_string(cell→getX() + 1) + std::to_string(cell→getY(
) + 1));
}

int Node::getG() const {
    return g;
}

void Node::setG(int g) {
    Node::g = g;
}

int Node::getH() const {
    return h;
}

void Node::setH(int h) {
    Node::h = h;
}

const std::shared_ptr<Cell> &Node::getCell() const {
    return cell;
}

int Node::getF() {
    return g + h;
}

int Node::getId() const {
    return id;
}

bool operator≡(const Node& node1, const Node &node2) {
    return node1.getCell() ≡ node2.getCell();
}

bool operator≠(const Node& node1, const Node &node2) {
    return ¬(node1 ≡ node2);
}

void Node::setParent(int aParent) {
    parent = aParent;
}

int Node::getParent() const {
    return parent;
}

Node::~Node() = default;
```

jul 21, 20 15:20 **NodeContainer.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_NODECONTAINER_H
#define ARGENTUM_TALLER_NODECONTAINER_H


#include <map>
#include "Node.h"

class NodeContainer {
private:
    std::map<int, Node> nodes;
public:
    NodeContainer();

    void insert(const Node& aNode);

    bool has(const Node& aNode);

    Node getBestNode();

    virtual ~NodeContainer();

    void modifyNode(int i, Node &node);

    Node &get(int i);
};


#endif //ARGENTUM_TALLER_NODECONTAINER_H
```

jul 21, 20 15:20 **NodeContainer.cpp** Page 1/1

```cpp
#include "NodeContainer.h"

NodeContainer::NodeContainer() = default;

NodeContainer::~NodeContainer() = default;

void NodeContainer::insert(const Node& aNode) {
    nodes.insert(std::pair<int, Node>(aNode.getId(), aNode));
}

bool NodeContainer::has(const Node& aNode) {
    bool insideContainer = true;
    try {
        nodes.at(aNode.getId());
    } catch (std::exception &e) {
        insideContainer = false;
    }
    return insideContainer;
}

Node NodeContainer::getBestNode() {
    auto iter = nodes.begin();
    int bestF = 0;
    Node aNode = (*iter).second;
    while (iter ≠ nodes.end()) {
        if (bestF ≡ 0 ∨ (*iter).second.getF() < bestF) {
            bestF = (*iter).second.getF();
            aNode = (*iter).second;
        }
        iter++;
    }
    nodes.erase(aNode.getId());
    return aNode;
}

void NodeContainer::modifyNode(int id, Node &aNode) {
    Node &innerNode = nodes.at(id);
    if (aNode.getG() + 1 < innerNode.getG()) {
        innerNode.setG(aNode.getG() + 1);
        innerNode.setParent(aNode.getId());
    }
}

Node &NodeContainer::get(int id) {
    return nodes.at(id);
}
```

jul 21, 20 15:20 **Nest.h** Page 1/1

```
1  #ifndef ARGENTUM_TALLER_NEST_H
2  #define ARGENTUM_TALLER_NEST_H
3
4
5  #include <vector>
6  #include <memory>
7  #include "Cell.h"
8
9  class Nest {
10 private:
11     uint8_t nestLimit;
12     uint nestId;
13     std::vector<std::shared_ptr<Cell>> nestCells;
14     std::vector<uint> creatures;
15
16 public:
17     explicit Nest(uint8_t nestLimit, uint nestId, std::vector<std::shared_ptr<Ce
   ll>> cells);
18
19     ~Nest();
20
21     void addCreature(uint id);
22
23     std::shared_ptr<Cell> getFreeCell();
24
25     friend bool operator<(const Nest& firstNest, const Nest& secondNest);
26
27     uint getNestId() const;
28
29     const std::vector<uint> &getCreatures() const;
30
31     int getAmountCreatures() const ;
32
33     void removeCreature(uint id);
34
35     bool isFull() const;
36 };
37
38
39 #endif //ARGENTUM_TALLER_NEST_H
```

jul 21, 20 15:20 **Nest.cpp** Page 1/1

```
1  #include "Nest.h"
2  #include <utility>
3
4  Nest::Nest(uint8_t nestLimit, uint nestId, std::vector<std::shared_ptr<Cell>> ce
   lls):
5  nestLimit(nestLimit), nestId(nestId), nestCells(std::move(cells)) {}
6
7  bool Nest::isFull() const {
8      bool isFull = creatures.size() ≡ nestLimit;
9      if (¬isFull) {
10         isFull = true;
11         for (auto &aCell : nestCells) {
12             if (aCell→isEmpty()) {
13                 isFull = false;
14                 break;
15             }
16         }
17     }
18     return isFull;
19 }
20
21 std::shared_ptr<Cell> Nest::getFreeCell() {
22     std::shared_ptr<Cell> returnedCell;
23     for (auto &cell : nestCells) {
24         if (cell→isEmpty()) {
25             returnedCell = cell;
26         }
27     }
28     return returnedCell;
29 }
30
31 void Nest::addCreature(uint id) {
32     creatures.push_back(id);
33 }
34
35 const std::vector<uint> &Nest::getCreatures() const {
36     return creatures;
37 }
38
39 uint Nest::getNestId() const {
40     return nestId;
41 }
42
43 void Nest::removeCreature(uint id) {
44     auto iter = creatures.begin();
45     while (iter ≠ creatures.end()) {
46         if ((*iter) ≡ id) {
47             iter = this→creatures.erase(iter);
48         } else {
49             iter++;
50         }
51     }
52 }
53
54 int Nest::getAmountCreatures() const {
55     return creatures.size();
56 }
57
58 bool operator<(const Nest& firstNest,const Nest &secondNest) {
59     return firstNest.getAmountCreatures() < secondNest.getAmountCreatures();
60 }
61
62
63 Nest::~Nest() = default;
```

jul 21, 20 15:20 **NestContainer.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_NESTCONTAINER_H
#define ARGENTUM_TALLER_NESTCONTAINER_H


#include <vector>
#include "Nest.h"

class NestContainer {
private:
    std::vector<Nest> nests;
    uint8_t nestIndex = 0;
    uint8_t length{};

    uint8_t getNextIndex();
public:
    NestContainer();

    explicit NestContainer(std::vector<Nest> nest);

    virtual ~NestContainer();

    Nest& getNextNestAvailable();

    std::vector<Nest> &getNests();

    int getAmountCreatures();

    Nest &getNest(uint i);
};


#endif //ARGENTUM_TALLER_NESTCONTAINER_H
```

jul 21, 20 15:20 **NestContainer.cpp** Page 1/1

```cpp
#include "NestContainer.h"
#include "../common/Exception.h"

#include <utility>
#include <algorithm>

NestContainer::NestContainer(std::vector<Nest> nests) : nests(std::move(nests))
{
    length = this→nests.size();
}

Nest& NestContainer::getNextNestAvailable() {
    std::sort_heap(nests.begin(), nests.end());
    for (auto &aNest : nests) {
        if (¬aNest.isFull() ) {
            return aNest;
        }
    }
    throw Exception("There is not available nests to add creatures");
}
//    uint8_t counter = 0;
//    Nest& nest = nests.at(getNextIndex());
//    while (nest.isFull() && counter < length) {
//        nest = nests.at(getNextIndex());
//        counter++;
//    }
//    if (nest.isFull()) {
//
//    }
//    return nest;


std::vector<Nest> &NestContainer::getNests() {
    return nests;
}

Nest &NestContainer::getNest(uint nestId) {
    size_t index = 0;
    for (auto &aNest : nests) {
        if (nestId ≡ aNest.getNestId()) {
            break;
        }
        index++;
    }
    return nests.at(index);
}

int NestContainer::getAmountCreatures() {
    int amountCreatures = 0;
    for (auto &aNest : nests) {
        amountCreatures += aNest.getAmountCreatures();
    }
    return amountCreatures;
}

uint8_t NestContainer::getNextIndex() {
    nestIndex++;
    nestIndex = nestIndex ≥ length ? 0 : nestIndex;
    return nestIndex;
}

NestContainer::NestContainer() = default;

NestContainer::~NestContainer() = default;
```

jul 21, 20 15:20 **Movement.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_MOVEMENT_H
#define ARGENTUM_TALLER_MOVEMENT_H


#include <cstdint>
#include "../common/Identificators.h"
#include "GameStatsConfig.h"

class Movement {
protected:
    bool finalized;
    bool initialized;
    Point firstPoint;
    Direction direction;
    float distance;
    float partialDistance;
public:
    Movement();

    bool isOver() const;

    bool hasStart() const;

    void reset();

    void stop();

    Point doStep();

    virtual float getAmountMovement() = 0;

    virtual ~Movement();

};


#endif //ARGENTUM_TALLER_MOVEMENT_H
```

jul 21, 20 15:20 **MovementCreature.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_MOVEMENTCREATURE_H
#define ARGENTUM_TALLER_MOVEMENTCREATURE_H


#include "Movement.h"

class MovementCreature : public Movement {
private:
    CreatureID creatureId = CreatureID::Nothing;
public:
    MovementCreature();

    void start(Point aFirstPoint, Direction aDirection, CreatureID creatureId);

    virtual ~MovementCreature();

    float getAmountMovement() override;
};


#endif //ARGENTUM_TALLER_MOVEMENTCREATURE_H
```

jul 21, 20 15:20 **MovementCreature.cpp** Page 1/1

```cpp
#include "MovementCreature.h"

MovementCreature::MovementCreature() : Movement() {}

MovementCreature::~MovementCreature() = default;

void MovementCreature::start(Point aFirstPoint, Direction aDirection, CreatureID aCreatureId) {
    creatureId = aCreatureId;
    partialDistance = 0.0f;
    distance = GameStatsConfig::getDistance();
    direction = aDirection;
    firstPoint = aFirstPoint;
    initialized = true;
}

float MovementCreature::getAmountMovement() {
    return GameStatsConfig::getAmountMovement(creatureId);
}
```

jul 21, 20 15:20 **Movement.cpp** Page 1/1

```cpp
#include "Movement.h"

Movement::Movement() : finalized(false), initialized(false), firstPoint(0.0,0.0),
    direction(Direction::down), distance(0), partialDistance(0) {}

Movement::~Movement() = default;

bool Movement::isOver() const {
    return finalized;
}

bool Movement::hasStart() const {
    return initialized;
}

void Movement::reset() {
    partialDistance = 0.0f;
    initialized = false;
    finalized = false;
}

Point Movement::doStep() {
    Point newPoint = firstPoint;
    partialDistance = partialDistance + getAmountMovement() > distance ? distance : partialDistance + getAmountMovement();
    switch(direction) {
        case Direction::up:
            newPoint.y -= partialDistance;
            break;
        case Direction::down:
            newPoint.y += partialDistance;
            break;
        case Direction::left:
            newPoint.x -= partialDistance;
            break;
        case Direction::right:
            newPoint.x += partialDistance;
            break;
    }
    if (partialDistance ≥  distance) {
        finalized = true;
    }
    return newPoint;
}

void Movement::stop() {
    finalized = true;
}
```

jul 21, 20 15:20 **MovementCharacter.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_MOVEMENTCHARACTER_H
#define ARGENTUM_TALLER_MOVEMENTCHARACTER_H


#include "Movement.h"

class MovementCharacter : public Movement {
private:
    RaceID raceId = RaceID::Nothing;
public:
    MovementCharacter();

    void start(Point aFirstPoint, Direction aDirection, RaceID raceId);

    float getAmountMovement() override;

    ~MovementCharacter() override;

};


#endif //ARGENTUM_TALLER_MOVEMENTCHARACTER_H
```

jul 21, 20 15:20 **MovementCharacter.cpp** Page 1/1

```cpp
#include "MovementCharacter.h"

MovementCharacter::MovementCharacter() : Movement() {}

MovementCharacter::~MovementCharacter() = default;

void MovementCharacter::start(Point aFirstPoint, Direction aDirection, RaceID aRaceId) {
    raceId = aRaceId;
    partialDistance = 0.0f;
    distance = GameStatsConfig::getDistance();
    direction = aDirection;
    firstPoint = aFirstPoint;
    initialized = true;
}

float MovementCharacter::getAmountMovement() {
    return GameStatsConfig::getAmountMovement(raceId);
}
```

jul 21, 20 15:20 **Merchant.h** Page 1/1

```
#ifndef MERCHANT_H
#define MERCHANT_H

#include <unordered_map>
#include "../common/Identificators.h"
#include "GameStats.h"
#include "Profession.h"
#include "GameCharacter.h"

//Se optÃ³ por realizar esta clase como un Singleton debido a que
//los diversos jugadores(Threads) no podrÃ¡n modificar esta clase
//ya que los items que se venden/compran no se alteran dentro de
//esta clase
class Merchant:public Profession {
private:
    Merchant();
    std::unordered_map<ItemsInventoryID, uint, std::hash<ItemsInventoryID>> items;

    static Merchant* merchant;

    ItemsInventoryID buyItem(ItemsInventoryID idItem, uint* balance) const;

    uint sellItem(ItemsInventoryID idItem) const;

    std::unordered_map<ItemsInventoryID,uint> getItems() const;
public:
    static Merchant* getInstance();

    void init(const std::map<ItemsInventoryID, ItemInfo>& itemsToInit);

    NPCInfo getInfo(uint id) override;

    void processInput(GameCharacter &character, InputInfo inputInfo) override;

    virtual ~Merchant();
};

#endif
```

jul 21, 20 15:20 **Merchant.cpp** Page 1/2

```
#include "Merchant.h"
#include "GameCharacter.h"

Merchant* Merchant::merchant = nullptr;

Merchant::Merchant() :items() {
    this→actions.push_back(ActionsProfessionID::Buy);
    this→actions.push_back(ActionsProfessionID::Sell);
}

Merchant* Merchant::getInstance() {
    if (merchant ≡ nullptr)
        merchant = new Merchant();
    return merchant;
}

void Merchant::init(const std::map<ItemsInventoryID, ItemInfo>&itemsToInit) {
    for (auto &iter: itemsToInit){
        this→items.insert({iter.first, iter.second.goldCost});
    }
}

ItemsInventoryID Merchant::buyItem(ItemsInventoryID idItem, uint* balance) const {
    uint itemPrice = items.at(idItem);
    if (*balance < itemPrice)
        return ItemsInventoryID::Nothing;
    *balance −= itemPrice;
    return idItem;
}

uint Merchant::sellItem(ItemsInventoryID idItem) const {
    if (ItemsInventoryID::Nothing ≡ idItem ) {
        return 0;
    }
    auto iter = items.find(idItem);
    return (*iter).second;
}

std::unordered_map<ItemsInventoryID,uint> Merchant::getItems() const {
    return items;
}

NPCInfo Merchant::getInfo(uint id) {
    NPCInfo info;
    info.type = 1;
    info.actions = actions;
    info.gold = 0;
    info.items = getItems();
    return info;
}

void Merchant::processInput(GameCharacter &character, InputInfo inputInfo) {
    uint sell, goldAmount;
    switch (inputInfo.input) {
        case InputID::sell:
            sell = sellItem(ItemsInventoryID(inputInfo.aditional));
            character.removeItemFromInventory(ItemsInventoryID(inputInfo.aditional));
            character.gainGold(sell);
            break;
        case InputID::buy:
            goldAmount = character.getGoldAmount();
            if (¬character.inventoryIsFull()) {
                ItemsInventoryID anItem = buyItem(ItemsInventoryID(inputInfo.aditional), &goldAmount);
```

jul 21, 20 15:20 **Merchant.cpp** Page 2/2

```
                if (anItem ≠ ItemsInventoryID::Nothing) {
                    character.addItemToInventory(anItem);
                    character.setGoldAmount(goldAmount);
                }
            }
            break;
        default:
            break;
    }
}

Merchant::~Merchant() {}
```

jul 21, 20 15:20 **ItemTranslator.h** Page 1/1

```
#ifndef ITEMTRANSLATOR_H
#define ITEMTRANSLATOR_H

#include "../common/Identificators.h"

class ItemTranslator {
public:
    ItemTranslator();

    static ItemsInventoryID weaponToItem(WeaponID weapon);
    static ItemsInventoryID shieldToItem(ShieldID shield);
    static ItemsInventoryID bodyToItem(BodyID body);
    static ItemsInventoryID helmetToItem(HelmetID helmet);

    static WeaponID itemToWeapon(ItemsInventoryID item);
    static ShieldID itemToShield(ItemsInventoryID item);
    static BodyID itemToBody(ItemsInventoryID item);
    static HelmetID itemToHelmet(ItemsInventoryID item);

    ~ItemTranslator();
};

#endif
```

jul 21, 20 15:20 **ItemTranslator.cpp** Page 1/4

```cpp
#include "ItemTranslator.h"

ItemTranslator::ItemTranslator() = default;

ItemsInventoryID ItemTranslator::weaponToItem(WeaponID weapon){
    ItemsInventoryID idItem = ItemsInventoryID::Nothing;
    switch(weapon) {
    case WeaponID::SimpleArc:
      idItem = ItemsInventoryID::SimpleArc;
      break;
    case WeaponID::CompoundArc:
      idItem = ItemsInventoryID::CompoundArc;
      break;
    case WeaponID::LongSword:
      idItem = ItemsInventoryID::LongSword;
      break;
    case WeaponID::Hammer:
      idItem = ItemsInventoryID::Hammer;
      break;
    case WeaponID::Ax:
      idItem = ItemsInventoryID::Ax;
      break;
    case WeaponID::ElficFlaute:
      idItem = ItemsInventoryID::ElficFlaute;
      break;
    case WeaponID::AshStick:
      idItem = ItemsInventoryID::AshStick;
      break;
    case WeaponID::Crosier:
      idItem = ItemsInventoryID::Crosier;
      break;
    case WeaponID::GnarledStick:
      idItem = ItemsInventoryID::GnarledStick;
      break;
    case WeaponID::Nothing:
      idItem = ItemsInventoryID::Nothing;
      break;
  }
    return idItem;
}

ItemsInventoryID ItemTranslator::bodyToItem(BodyID body){
    ItemsInventoryID idItem = ItemsInventoryID::Nothing;
    switch(body) {
    case BodyID::BlueCommon:
      idItem = ItemsInventoryID::BlueCommon;
      break;
    case BodyID::GreenCommon:
      idItem = ItemsInventoryID::GreenCommon;
      break;
    case BodyID::RedCommon:
      idItem = ItemsInventoryID::RedCommon;
      break;
    case BodyID::LeatherArmor:
      idItem = ItemsInventoryID::LeatherArmor;
      break;
    case BodyID::BlueTunic:
      idItem = ItemsInventoryID::BlueTunic;
      break;
    case BodyID::PlateArmor:
      idItem = ItemsInventoryID::PlateArmor;
      break;
    default:
      idItem = ItemsInventoryID::Nothing;
      break;
    }
```

jul 21, 20 15:20 **ItemTranslator.cpp** Page 2/4

```cpp
    return idItem;
}

ItemsInventoryID ItemTranslator::helmetToItem(HelmetID helmet){
    ItemsInventoryID idItem = ItemsInventoryID::Nothing;
    switch(helmet) {
    case HelmetID::Hood:
      idItem = ItemsInventoryID::Hood;
      break;
    case HelmetID::MagicHat:
      idItem = ItemsInventoryID::MagicHat;
      break;
    case HelmetID::IronHelmet:
      idItem = ItemsInventoryID::IronHelmet;
      break;
    case HelmetID::Nothing:
      idItem = ItemsInventoryID::Nothing;
      break;
    default:
      idItem = ItemsInventoryID::Nothing;
      break;
  }
    return idItem;
}

ItemsInventoryID ItemTranslator::shieldToItem(ShieldID shield){
    ItemsInventoryID idItem = ItemsInventoryID::Nothing;
  switch(shield) {
    case ShieldID::IronShield:
      idItem = ItemsInventoryID::IronShield;
      break;
    case ShieldID::TurtleShield:
      idItem = ItemsInventoryID::TurtleShield;
      break;
    case ShieldID::Nothing:
      idItem = ItemsInventoryID::Nothing;
      break;
  }
    return idItem;
}

WeaponID ItemTranslator::itemToWeapon(ItemsInventoryID item){
    WeaponID weapon;
    switch(item) {
    case ItemsInventoryID::SimpleArc:
      weapon = WeaponID::SimpleArc;
      break;
    case ItemsInventoryID::CompoundArc:
      weapon = WeaponID::CompoundArc;
      break;
    case ItemsInventoryID::LongSword:
      weapon = WeaponID::LongSword;
      break;
    case ItemsInventoryID::Hammer:
      weapon = WeaponID::Hammer;
      break;
    case ItemsInventoryID::Ax:
      weapon = WeaponID::Ax;
      break;
    case ItemsInventoryID::ElficFlaute:
      weapon = WeaponID::ElficFlaute;
      break;
    case ItemsInventoryID::AshStick:
      weapon = WeaponID::AshStick;
      break;
    case ItemsInventoryID::Crosier:
```

jul 21, 20 15:20 **ItemTranslator.cpp** Page 3/4

```
133         weapon = WeaponID::Crosier;
134         break;
135       case ItemsInventoryID::GnarledStick:
136         weapon = WeaponID::GnarledStick;
137         break;
138           default:
139               weapon = WeaponID::Nothing;
140       }
141       return weapon;
142   }
143
144   ShieldID ItemTranslator::itemToShield(ItemsInventoryID item){
145       ShieldID shield = ShieldID::Nothing;
146       switch (item) {
147           case ItemsInventoryID::IronShield:
148         shield = ShieldID::IronShield;
149         break;
150       case ItemsInventoryID::TurtleShield:
151         shield = ShieldID::TurtleShield;
152         break;
153           default:
154               shield = ShieldID::Nothing;
155       }
156       return shield;
157   }
158
159   BodyID ItemTranslator::itemToBody(ItemsInventoryID item){
160       BodyID body = BodyID::Nothing;
161       switch (item) {
162           case ItemsInventoryID::BlueCommon:
163         body = BodyID::BlueCommon;
164         break;
165       case ItemsInventoryID::GreenCommon:
166         body = BodyID::GreenCommon;
167         break;
168       case ItemsInventoryID::RedCommon:
169         body = BodyID::RedCommon;
170         break;
171       case ItemsInventoryID::LeatherArmor:
172         body = BodyID::LeatherArmor;
173         break;
174       case ItemsInventoryID::BlueTunic:
175         body = BodyID::BlueTunic;
176         break;
177       case ItemsInventoryID::PlateArmor:
178         body = BodyID::PlateArmor;
179         break;
180       default:
181         body = BodyID::Nothing;
182         break;
183       }
184       return body;
185   }
186
187   HelmetID ItemTranslator::itemToHelmet(ItemsInventoryID item){
188       HelmetID helmet = HelmetID::Nothing;
189       switch (item) {
190           case ItemsInventoryID::Hood:
191         helmet = HelmetID::Hood;
192         break;
193       case ItemsInventoryID::MagicHat:
194         helmet = HelmetID::MagicHat;
195         break;
196       case ItemsInventoryID::IronHelmet:
197         helmet = HelmetID::IronHelmet;
198         break;
```

jul 21, 20 15:20 **ItemTranslator.cpp** Page 4/4

```
199           case ItemsInventoryID::Nothing:
200               helmet = HelmetID::Nothing;
201               break;
202       default:
203         helmet = HelmetID::Nothing;
204         break;
205       }
206       return helmet;
207   }
208
209   ItemTranslator::~ItemTranslator() = default;
```

jul 21, 20 15:20 **Inventory.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_INVENTORY_H
#define ARGENTUM_TALLER_INVENTORY_H


#include "../common/Identificators.h"
#include <vector>

class Inventory {
private:
    std::vector<ItemsInventoryID> inventoryItems;
    uint8_t itemsAmount;
    uint8_t limit;
public:
    Inventory();

    ItemsInventoryID getItem(int index) const;


    bool addItem(ItemsInventoryID aItemInventoryId);

    bool isEmpty();

    bool isFull() const;

    const std::vector<ItemsInventoryID> &getInventoryItems() const;

    ItemsInventoryID removeItem(ItemsInventoryID aItemToRemove);

    std::string getStringInventory() const;

    void clear();

    virtual ~Inventory();
};


#endif //ARGENTUM_TALLER_INVENTORY_H
```

jul 21, 20 15:20 **Inventory.cpp** Page 1/2

```
#include "Inventory.h"

#include <utility>
#include <algorithm>
#include "GameStatsConfig.h"

Inventory::Inventory() :
inventoryItems(), itemsAmount(0),
limit(GameStatsConfig::getInventoryLimit()){
    for (int i = 0; i < limit; ++i){
        inventoryItems.push_back(ItemsInventoryID::Nothing);
    }
}

ItemsInventoryID Inventory::getItem(int index) const {
    return inventoryItems.at(index);
}

bool Inventory::addItem(ItemsInventoryID aItemInventoryId) {
    bool added = false;
    if (¬isFull() ∧ aItemInventoryId ≠ ItemsInventoryID::Nothing) {
        auto iter = std::find(inventoryItems.begin(), inventoryItems.end(), Item
sInventoryID::Nothing);
        (*iter) = aItemInventoryId;
        itemsAmount++;
        added = true;
    }
    return added;
}

bool Inventory::isFull() const {
    return itemsAmount ≡ limit;
}

void Inventory::clear() {
    inventoryItems.clear();
    itemsAmount = 0;
    for (int i = 0; i < limit; ++i){
        inventoryItems.push_back(ItemsInventoryID::Nothing);
    }
}

std::string Inventory::getStringInventory() const {
    std::string inv;
    std::string temp;
    for (int i = 0; i < limit; ++i){
        temp = std::to_string((int)inventoryItems.at(i));
        if (temp.size() ≡ 1)
            inv += "0";
        inv += temp;
        if (i ≠ limit-1)
            inv += "|";
    }
    return inv;
}

ItemsInventoryID Inventory::removeItem(ItemsInventoryID aItemToRemove) {
    auto iter = std::find(inventoryItems.begin(), inventoryItems.end(), aItemToR
emove);
    if (iter ≡ inventoryItems.end() ∨ aItemToRemove ≡ ItemsInventoryID::Nothing)
 {
        return ItemsInventoryID::Nothing;
    }
    (*iter) = ItemsInventoryID::Nothing;
    itemsAmount--;
    return aItemToRemove;
```

jul 21, 20 15:20 **Inventory.cpp** Page 2/2

```
64  }
65
66  const std::vector<ItemsInventoryID> &Inventory::getInventoryItems() const {
67      return inventoryItems;
68  }
69
70  bool Inventory::isEmpty() {
71      return itemsAmount == 0;
72  }
73
74  Inventory::~Inventory() = default;
```

jul 21, 20 15:20 **GameStats.h** Page 1/1

```
1   #ifndef ARGENTUM_TALLER_GAMESTATS_H
2   #define ARGENTUM_TALLER_GAMESTATS_H
3
4   #include <cstdint>
5
6   struct RaceInfo {
7       float intelligent;
8       float recoveryTime;
9       float health;
10      float mana;
11      float constitution;
12      float strength;
13      float agility;
14  };
15
16  struct GameClassInfo {
17      float health;
18      float mana;
19      float meditation;
20  };
21
22  struct ItemInfo {
23      uint8_t id;
24      std::string name;
25      bool damage;
26      uint8_t range;
27      uint8_t minDamage;
28      uint8_t maxDamage;
29      uint8_t manaUsed;
30      uint8_t manaRestored;
31      uint8_t healthRestored;
32      uint8_t minDefense;
33      uint8_t maxDefense;
34      uint8_t goldCost;
35      std::string type;
36  };
37
38  struct CreatureInfo {
39      float health;
40      float strength;
41      float agility;
42      float constitution;
43  };
44
45  #endif //ARGENTUM_TALLER_GAMESTATS_H
```

jul 21, 20 15:20 **GameStatsConfig.h** Page 1/2

```
#ifndef ARGENTUM_TALLER_GAMESTATSCONFIG_H
#define ARGENTUM_TALLER_GAMESTATSCONFIG_H

#include <unordered_map>
#include "GameStats.h"
#include "../common/Identificators.h"
#include <rapidjson/document.h>
#include <map>

class GameStatsConfig {
private:
    static std::unordered_map<RaceID, RaceInfo, std::hash<RaceID>> races;
    static std::unordered_map<GameClassID, GameClassInfo, std::hash<GameClassID>
> gameClasses;
    static std::map<ItemsInventoryID, ItemInfo> items;
    static std::unordered_map<CreatureID, CreatureInfo, std::hash<CreatureID>> c
reatures;
    static std::string port;
    static float goldRandMin;
    static float goldRandMax;
    static float goldMaxMult;
    static float goldMaxPot;
    static float expMaxMult;
    static float expMaxPot;
    static float evadeRandMin;
    static float evadeRandMax;
    static float evadeProbability;
    static float expRandMin;
    static float expRandMax;
    static uint8_t levelDifference;
    static float minAgility;
    static uint8_t creaturesLimit;
    static uint8_t nestCreaturesLimit;
    static float distance;
    static int inventoryLimit;
    static int newbieLevel;
    static float loseExp;
    RaceInfo createRaceInfo(rapidjson::Value &value);
    GameClassInfo createGameClass(rapidjson::Value &value);
    ItemInfo createItem(rapidjson::Value& value);
    CreatureInfo createCreatureInfo(rapidjson::Value &value);
public:
    GameStatsConfig();

    explicit GameStatsConfig(rapidjson::Document &json);

    virtual ~GameStatsConfig();
    static std::string getPort();
    static float getMaxHealth(RaceID raceId, GameClassID gameClassId, uint level
);
    static float getMaxHealth(CreatureID creatureId, uint level);
    static float getRecoveryHealth(RaceID raceId);
    static float getMaxMana(RaceID raceId, GameClassID gameClassId, uint level);
    static float getRecoveryMana(RaceID raceId);
    static float getRecoveryManaMeditation(RaceID raceId, GameClassID gameClassI
d);
    static float getGoldDrop(CreatureID creatureId, uint level);
    static float getMaxGold(uint level);
    static float getNextLevelLimit(uint level);
    static float getExp(float damage, uint level, uint enemyLevel);
    static float getAdditionalExp(float damage, float enemyMaxLife, uint level,
uint enemyLevel);
    static float getDamage(RaceID race, WeaponID weaponId);
    static float getDamage(CreatureID creatureId);
    static bool canEvade(RaceID raceId);
    static bool canEvade(CreatureID creatureId);
```

jul 21, 20 15:20 **GameStatsConfig.h** Page 2/2

```
    static float getDefense(BodyID bodyId, ShieldID shieldId, HelmetID helmetId)
;
    static float getDefense(CreatureID creatureId);
    static std::map<ItemsInventoryID,ItemInfo> getItems();
    static uint8_t getAmountMovement(RaceID raceId);
    static uint8_t getAmountMovement(CreatureID creatureId);
    static uint8_t getCreaturesLimit();
    static const ItemInfo getItem(ItemsInventoryID id);
    static uint8_t getNestCreatureLimit();
    static float getDistance();
    static int getInventoryLimit();
    static int getWeaponDistance(WeaponID aWeaponId);
    static int getWeaponCost(WeaponID aWeaponId);
    static bool canAttack(int level, int enemyLevel);
    static bool isNewbie(int level);
    static float restoreHealth(WeaponID aWeaponId);
    static float getLoseExp();
};


#endif //ARGENTUM_TALLER_GAMESTATSCONFIG_H
```

```cpp
#include "GameStatsConfig.h"
#include "../common/Random.h"
#include "ItemTranslator.h"
#include <iostream>

GameStatsConfig::GameStatsConfig(rapidjson::Document &json) {
    GameStatsConfig gameStatsConfig;
    port = json["port"].GetString();
    goldRandMin = json["goldRandMin"].GetFloat();
    goldRandMax = json["goldRandMax"].GetFloat();
    goldMaxMult = json["goldMaxMult"].GetFloat();
    goldMaxPot = json["goldMaxPot"].GetFloat();
    expRandMin = json["expRandMin"].GetFloat();
    expRandMax = json["expRandMax"].GetFloat();
    expMaxMult = json["expMaxMult"].GetFloat();
    expMaxPot = json["expMaxPot"].GetFloat();
    evadeRandMin = json["evadeRandMin"].GetFloat();
    evadeRandMax = json["evadeRandMax"].GetFloat();
    evadeProbability = json["evadeProbability"].GetFloat();
    levelDifference = json["levelDifference"].GetFloat();
    minAgility = json["minAgility"].GetFloat();
    creaturesLimit = json["creaturesLimit"].GetInt();
    nestCreaturesLimit = json["nestCreaturesLimit"].GetInt();
    distance = json["distance"].GetFloat();
    inventoryLimit = json["inventoryLimit"].GetInt();
    newbieLevel = json["newbieLevel"].GetInt();
    loseExp = json["loseExp"].GetFloat();

    rapidjson::Value::Array racesArray = json["races"].GetArray();
    for (auto &aRace : racesArray) {
        races.insert(std::pair<RaceID, RaceInfo>(RaceID(aRace["id"].GetInt()), c
reateRaceInfo(aRace)));
    }
    rapidjson::Value::Array gameClassesArray = json["classes"].GetArray();
    for (auto &aGameClass : gameClassesArray) {
        gameClasses.insert(std::pair<GameClassID, GameClassInfo>(GameClassID(aGa
meClass["id"].GetInt()), createGameClass(aGameClass)));
    }

    rapidjson::Value::Array itemsArray = json["items"].GetArray();
    for (auto &aItem: itemsArray) {
        items.insert(std::pair<ItemsInventoryID, ItemInfo>(ItemsInventoryID(aIte
m["id"].GetInt()), createItem(aItem)));
    }

    rapidjson::Value::Array creatuesArray = json["creatures"].GetArray();
    for (auto &aCreature : creatuesArray) {
        creatures.insert(std::pair<CreatureID, CreatureInfo>(CreatureID(aCreatur
e["id"].GetInt()), createCreatureInfo(aCreature)));
    }
}

RaceInfo GameStatsConfig::createRaceInfo(rapidjson::Value &value) {
    RaceInfo aRaceInfo{};
    aRaceInfo.intelligent = value["intelligent"].GetFloat();
    aRaceInfo.recoveryTime = value["recoveryTime"].GetFloat();
    aRaceInfo.constitution = value["constitution"].GetFloat();
    aRaceInfo.strength = value["strength"].GetFloat();
    aRaceInfo.agility = value["agility"].GetFloat();
    aRaceInfo.health = value["health"].GetFloat();
    aRaceInfo.mana = value["mana"].GetFloat();
    return aRaceInfo;
}

GameClassInfo GameStatsConfig::createGameClass(rapidjson::Value &value) {
    GameClassInfo aGameClassInfo{};
```



```cpp
    aGameClassInfo.health = value["health"].GetFloat();
    aGameClassInfo.mana = value["mana"].GetFloat();
    aGameClassInfo.meditation = value["meditation"].GetFloat();
    return aGameClassInfo;
}

ItemInfo GameStatsConfig::createItem(rapidjson::Value& value) {
    ItemInfo aItemInfo{};
    aItemInfo.name = value["name"].GetString();
    aItemInfo.damage = value["damage"].GetBool();
    aItemInfo.minDamage = value["minDamage"].GetInt();
    aItemInfo.maxDamage = value["maxDamage"].GetInt();
    aItemInfo.manaUsed = value["manaUsed"].GetInt();
    aItemInfo.manaRestored = value["manaRestored"].GetInt();
    aItemInfo.healthRestored = value["healthRestored"].GetInt();
    aItemInfo.minDefense = value["minDefense"].GetInt();
    aItemInfo.maxDefense = value["maxDefense"].GetInt();
    aItemInfo.goldCost = value["goldCost"].GetInt();
    aItemInfo.type = value["type"].GetString();
    aItemInfo.range = value["range"].GetInt();
    return aItemInfo;
}

CreatureInfo GameStatsConfig::createCreatureInfo(rapidjson::Value& value) {
    CreatureInfo aCreatureInfo{};
    aCreatureInfo.strength = value["strength"].GetFloat();
    aCreatureInfo.agility = value["agility"].GetFloat();
    aCreatureInfo.health = value["health"].GetFloat();
    aCreatureInfo.constitution = value["constitution"].GetFloat();
    return aCreatureInfo;
}

const ItemInfo GameStatsConfig::getItem(ItemsInventoryID id) {
    return items.at(id);
}

float GameStatsConfig::getMaxHealth(RaceID raceId, GameClassID gameClass, uint l
evel) {
    float max = races.at(raceId).constitution * gameClasses.at(gameClass).health
 * races.at(raceId).health * level;
    return max;
}

float GameStatsConfig::getRecoveryHealth(RaceID raceId) {
    return races.at(raceId).recoveryTime/ (45 * 10);
}

float GameStatsConfig::getMaxMana(RaceID raceId, GameClassID gameClass, uint lev
el) {
    float max = races.at(raceId).intelligent * races.at(raceId).mana * gameClass
es.at(gameClass).mana * level;
    return max;
}

float GameStatsConfig::getRecoveryMana(RaceID race) {
    return races.at(race).recoveryTime / (45 * 10);
}

float GameStatsConfig::getRecoveryManaMeditation(RaceID race, GameClassID gameCl
ass) {
    return (races.at(race).intelligent * gameClasses.at(gameClass).meditation) /
 (45 * 10);
}

float GameStatsConfig::getGoldDrop(CreatureID creatureId, uint level){
    return Random::getFloat(goldRandMin, goldRandMax) * GameStatsConfig::getMaxH
```

```
ealth(creatureId, level);
}

float GameStatsConfig::getMaxGold(uint level){
    return goldMaxMult * pow(level, goldMaxPot);
}

float GameStatsConfig::getNextLevelLimit(uint level){
    return expMaxMult * pow(level, expMaxPot);
}

float GameStatsConfig::getExp(float damage, uint level, uint enemyLevel) {
    return damage * std::max((int)(enemyLevel - level + levelDifference), 0);
}

float GameStatsConfig::getAdditionalExp(float damage, float enemyMaxLife , uint level, uint enemyLevel){
    return Random::getFloat(expRandMin, expRandMax) * enemyMaxLife * std::max((int)(enemyLevel - level + levelDifference), 0);
}

float GameStatsConfig::getDamage(RaceID raceId, WeaponID weaponId){
    RaceInfo aRaceInfo = races.at(raceId);
    if (weaponId == WeaponID::Nothing) {
        return aRaceInfo.strength;
    }
    ItemInfo aWeapon = items.at(ItemTranslator::weaponToItem(weaponId));
    return aRaceInfo.strength * Random::get(aWeapon.minDamage, aWeapon.maxDamage);
}

bool GameStatsConfig::canEvade(RaceID race){
    double base = Random::getFloat(evadeRandMin,evadeRandMax);
    return pow(base, races.at(race).agility) < evadeProbability;
}

uint8_t GameStatsConfig::getAmountMovement(RaceID raceId) {
    return std::max(races.at(raceId).agility * 1.5f, minAgility);
}

uint8_t GameStatsConfig::getCreaturesLimit() {
    return creaturesLimit;
}

uint8_t GameStatsConfig::getNestCreatureLimit() {
    return nestCreaturesLimit;
}

std::string GameStatsConfig::getPort() {
    return port;
}

std::map<ItemsInventoryID,ItemInfo> GameStatsConfig::getItems() {
    return items;
}

float GameStatsConfig::getDistance(){
    return distance;
}

uint8_t GameStatsConfig::getAmountMovement(CreatureID creatureId) {
    return std::max(creatures.at(creatureId).agility, minAgility) / 2;
}

float GameStatsConfig::getDamage(CreatureID creatureId){
    return creatures.at(creatureId).strength;
```



```
}

float GameStatsConfig::getDefense(BodyID bodyId, ShieldID shieldId, HelmetID helmetId) {
    float defense = 0.0;
    ItemInfo aItemInfo;
    if (bodyId ≠ BodyID::Nothing) {
        aItemInfo = items.at(ItemTranslator::bodyToItem(bodyId));
        defense += Random::get(aItemInfo.minDefense, aItemInfo.maxDefense);
    }
    if (shieldId ≠ ShieldID::Nothing) {
        aItemInfo = items.at(ItemTranslator::shieldToItem(shieldId));
        defense += Random::get(aItemInfo.minDefense, aItemInfo.maxDefense);
    }
    if (helmetId ≠ HelmetID::Nothing) {
        aItemInfo = items.at(ItemTranslator::helmetToItem(helmetId));
        defense += Random::get(aItemInfo.minDefense, aItemInfo.maxDefense);
    }
    return defense;
}

float GameStatsConfig::getDefense(CreatureID creatureId) {
    return Random::getFloat(0.0, creatures.at(creatureId).constitution);
}

bool GameStatsConfig::canEvade(CreatureID creatureId) {
    double base = Random::getFloat(evadeRandMin,evadeRandMax);
    return pow(base, creatures.at(creatureId).agility) < evadeProbability;
}

float GameStatsConfig::getMaxHealth(CreatureID creatureId, uint level) {
    return creatures.at(creatureId).health * level;
}

int GameStatsConfig::getInventoryLimit() {
    return GameStatsConfig::inventoryLimit;
}

int GameStatsConfig::getWeaponDistance(WeaponID aWeaponId) {
    if ( aWeaponId ≡ WeaponID::Nothing) {
        return 1;
    }
    ItemInfo aItemInfo = items.at(ItemTranslator::weaponToItem(aWeaponId));
    return aItemInfo.range;
}

int GameStatsConfig::getWeaponCost(WeaponID aWeaponId) {
    return items.at(ItemTranslator::weaponToItem(aWeaponId)).manaUsed;
}

bool GameStatsConfig::canAttack(int level, int enemyLevel) {
    if (isNewbie(level) ∨ isNewbie(enemyLevel))
        return false;
    int diff = level - enemyLevel;
    if (diff < -GameStatsConfig::levelDifference ∨ diff > GameStatsConfig::levelDifference)
        return false;
    return true;
}

bool GameStatsConfig::isNewbie(int level){
    return level ≤ GameStatsConfig::newbieLevel;
}

float GameStatsConfig::restoreHealth(WeaponID aWeaponId) {
    if ( aWeaponId ≡ WeaponID::Nothing) {
```

jul 21, 20 15:20 **GameStatsConfig.cpp** Page 5/5

```
249         return 0;
250     }
251     ItemInfo aItemInfo = items.at(ItemTranslator::weaponToItem(aWeaponId));
252     return aItemInfo.healthRestored;
253 }
254
255 float GameStatsConfig::getLoseExp() {
256     return loseExp;
257 }
258
259 GameStatsConfig::GameStatsConfig() = default;
260
261 GameStatsConfig::~GameStatsConfig() = default;
262
263 std::unordered_map<RaceID, RaceInfo, std::hash<RaceID>> GameStatsConfig::races;
264 std::unordered_map<GameClassID, GameClassInfo, std::hash<GameClassID>> GameStats
    Config::gameClasses;
265 std::map<ItemsInventoryID, ItemInfo> GameStatsConfig::items;
266 std::unordered_map<CreatureID, CreatureInfo, std::hash<CreatureID>> GameStatsCon
    fig::creatures;
267 std::string GameStatsConfig::port{};
268 float GameStatsConfig::goldRandMin = 0.0;
269 float GameStatsConfig::goldRandMax = 0.0;
270 float GameStatsConfig::goldMaxMult = 0.0;
271 float GameStatsConfig::goldMaxPot = 0.0;
272 float GameStatsConfig::expMaxMult = 0.0;
273 float GameStatsConfig::expMaxPot = 0.0;
274 float GameStatsConfig::evadeRandMin = 0.0;
275 float GameStatsConfig::evadeRandMax = 0.0;
276 float GameStatsConfig::evadeProbability = 0.0;
277 float GameStatsConfig::expRandMin = 0.0;
278 float GameStatsConfig::expRandMax = 0.0;
279 uint8_t GameStatsConfig::levelDifference = 0;
280 float GameStatsConfig::minAgility = 0.0;
281 uint8_t GameStatsConfig::creaturesLimit = 0.0;
282 uint8_t GameStatsConfig::nestCreaturesLimit = 0.0;
283 float GameStatsConfig::distance = 0.0;
284 int GameStatsConfig::inventoryLimit = 0;
285 int GameStatsConfig::newbieLevel = 0;
286 float GameStatsConfig::loseExp = 0;
```

jul 21, 20 15:20 **GameObjectsContainer.h** Page 1/1

```
1  #ifndef ARGENTUM_TALLER_GAMEOBJECTSCONTAINER_H
2  #define ARGENTUM_TALLER_GAMEOBJECTSCONTAINER_H
3
4
5  #include <memory>
6  #include <unordered_map>
7  #include "GameObject.h"
8
9  class GameObjectsContainer {
10 private:
11     std::unordered_map<uint, std::shared_ptr<GameObject>, std::hash<uint>> gameO
   bjects;
12 public:
13     GameObjectsContainer();
14
15     void addGameObject(std::shared_ptr<GameObject> aGameObject, uint id);
16
17     std::vector<GameObjectInfo> getUpdatedGameObjectsInfo();
18
19     std::vector<std::shared_ptr<GameObject>> getUpdatedGameObjects();
20
21     void update(Board& board);
22
23     std::shared_ptr<GameObject> getGameObject(uint id);
24
25     virtual ~GameObjectsContainer();
26
27     void deleteGameObject(const uint id, Board &board);
28
29     void removeCreaturesAndItems(Board& board);
30 };
31 #endif //ARGENTUM_TALLER_GAMEOBJECTSCONTAINER_H
```

jul 21, 20 15:20 **GameObjectsContainer.cpp** Page 1/1

```cpp
#include <iostream>
#include "GameObjectsContainer.h"

GameObjectsContainer::GameObjectsContainer() = default;

GameObjectsContainer::~GameObjectsContainer() = default;

void GameObjectsContainer::addGameObject(std::shared_ptr<GameObject> aGameObject, uint id) {
    gameObjects.insert(std::pair<uint, std::shared_ptr<GameObject>>(id, aGameObject));
}

std::vector<GameObjectInfo> GameObjectsContainer::getUpdatedGameObjectsInfo() {
    std::vector<GameObjectInfo> gameObjectsInfo;
    for (auto& gameObjectPair : gameObjects) {
        gameObjectsInfo.push_back(gameObjectPair.second→getGameObjectInfo());
    }
    return gameObjectsInfo;
}

std::vector<std::shared_ptr<GameObject>> GameObjectsContainer::getUpdatedGameObjects() {
    std::vector<std::shared_ptr<GameObject>> objects;
    for (auto& gameObjectPair : gameObjects) {
        objects.push_back(gameObjectPair.second);
    }
    return objects;
}


void GameObjectsContainer::update(Board& board) {
    for (auto& gameObjectPair : gameObjects) {
        gameObjectPair.second→update(gameObjects, board);
    }
}

std::shared_ptr<GameObject> GameObjectsContainer::getGameObject(uint id) {
    return gameObjects.at(id);
}

void GameObjectsContainer::deleteGameObject(uint id, Board &board) {
    this→gameObjects.at(id)→remove(board);
    this→gameObjects.erase(id);
}

void GameObjectsContainer::removeCreaturesAndItems(Board &board) {
    auto iter = this→gameObjects.begin();
    while (iter ≠ this→gameObjects.end()) {
        if ((*iter).second→isReadyToRemove()) {
            (*iter).second→remove(board);
            iter = this→gameObjects.erase(iter);
        } else {
            iter++;
        }
    }
}
```

jul 21, 20 15:20 **GameObject.h** Page 1/2

```cpp
#ifndef OBJETOJUEGO_H
#define OBJETOJUEGO_H

#include <string>
#include "../common/Identificators.h"
#include "../common/GameObjectInfo.h"
#include "GameStatsConfig.h"
#include "Board.h"
#include "DropItem.h"

class GameObject{
protected:
    Point point;
    std::shared_ptr<Cell> cell;
    uint id{};
    std::string textureHashId;
    Direction direction;
    NPCInfo infoInteracting;
    uint level;
    WeaponID interactWeapon;
public:
  explicit GameObject(uint id, Point initialPoint, std::shared_ptr<Cell> initialCell, Direction aDirection = Direction::down);

    GameObjectInfo getGameObjectInfo();

    virtual PlayerInfo getPlayerInfo();

    uint getId() const;

    void setDirection(Direction direction);

    void setCell(std::shared_ptr<Cell> aCell);

    void setPoint(Point aPoint);

    virtual bool isItem() = 0;

    virtual bool canDropsItems() = 0;

    virtual std::vector<DropItem> getDrop() = 0;

    void setInteractWeapon(WeaponID interactWeapon);

    std::shared_ptr<Cell> &getActualCell();

    void setTextureHashId(const std::string &textureHashId);

    virtual float getMaxLife() = 0;

    uint getLevel();

    virtual void update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board& board) = 0;

    virtual CharacterStateID getStateId() = 0;

    virtual bool isDead() = 0;

    virtual void receiveDamage(float damage, WeaponID weaponId) = 0;

   virtual NPCInfo interact(GameObject& character, InputInfo input) = 0;

    NPCInfo getInteractInfo() const;

    void setInteractInfo(NPCInfo info);
```

jul 21, 20 15:20 **GameObject.h** Page 2/2

```
65
66      virtual bool isReadyToRemove() = 0;
67
68      virtual void remove(Board &board) = 0;
69
70      virtual bool canBeAttacked(int enemyLevel) const = 0;
71
72      virtual bool hasAnInputInfo() = 0;
73
74      virtual InputInfo getNextInputInfo() = 0;
75
76      ~GameObject();
77  };
78
79  #endif
```

jul 21, 20 15:20 **GameObject.cpp** Page 1/1

```
1   #include "GameObject.h"
2
3   #include <utility>
4
5   GameObjectInfo GameObject::getGameObjectInfo() {
6       return GameObjectInfo(id, point, textureHashId, direction,getStateId(), isIt
    em(), interactWeapon);
7   }
8
9   uint GameObject::getId() const {
10      return id;
11  }
12
13  void GameObject::setDirection(Direction direction) {
14      GameObject::direction = direction;
15  }
16
17  void GameObject::setTextureHashId(const std::string &textureHashId) {
18      GameObject::textureHashId = textureHashId;
19  }
20
21  GameObject::GameObject(uint id, Point initialPoint, std::shared_ptr<Cell> initia
    lCell, Direction aDirection) :
22      point(initialPoint), cell(std::move(initialCell)), id(id), textureHashId(),
    direction(aDirection),
23      infoInteracting(), level(1), interactWeapon(WeaponID::Nothing) {}
24
25  std::shared_ptr<Cell> &GameObject::getActualCell() {
26      return cell;
27  }
28
29  void GameObject::setCell(std::shared_ptr<Cell> aCell) {
30      cell = std::move(aCell);
31  }
32
33  void GameObject::setPoint(Point aPoint) {
34      point = aPoint;
35  }
36
37  NPCInfo GameObject::getInteractInfo() const {
38      return this→infoInteracting;
39  }
40
41  void GameObject::setInteractInfo(NPCInfo info) {
42      this→infoInteracting = info;
43  }
44
45  uint GameObject::getLevel() {
46      return level;
47  }
48
49  PlayerInfo GameObject::getPlayerInfo() {
50      return PlayerInfo();
51  }
52
53  void GameObject::setInteractWeapon(WeaponID interactWeapon) {
54      GameObject::interactWeapon = interactWeapon;
55  }
56
57  GameObject::~GameObject()= default;
```

jul 21, 20 15:20 **GameCharacter.h** Page 1/2

```
#ifndef PERSONAJE_H
#define PERSONAJE_H

#include "../common/PlayerInfo.h"
#include "../common/StaticObject.h"
#include "../common/InputQueue.h"
#include "../common/Identificators.h"
#include "Inventory.h"
#include "GameObject.h"
#include <vector>
#include <memory>
#include <states/StatePoolCharacter.h>

class GameCharacter : public GameObject{
private:
    RaceID race{RaceID::Nothing};
    GameClassID gameClass{GameClassID::Nothing};
    uint goldAmount;
    float life;
    float mana;
    float exp;
    StatePoolCharacter statePool;
    InputQueue queueInputs;
    Inventory inventory;
    WeaponID weapon{WeaponID::Nothing};
    ShieldID shield{ShieldID::Nothing};
    HelmetID helmet{HelmetID::Nothing};
    BodyID body{BodyID::Nothing};
    ItemsInventoryID itemToDrop = ItemsInventoryID::Nothing;

    std::string updateTextureHashId();
    void consumePotion(const ItemInfo& potion);

public:
  GameCharacter(uint id, RaceID aRace, GameClassID aClass, std::shared_ptr<Cell>
 initialCell, Point initialPoint);

    PlayerInfo getPlayerInfo() override;

  void consumeMana();

    void upLevel();

    bool canUseWeapon();

    bool hasAnInputInfo() override;

    InputInfo getNextInputInfo() override;

    bool restoreHealth();

    bool inventoryIsFull();

  bool addItemToInventory(ItemsInventoryID aItemInventoryId);

  ItemsInventoryID removeItemFromInventory(ItemsInventoryID aItemToFind);

  void gainGold(int aGoldAmount);

    bool isReadyToRemove() override;

    WeaponID getWeapon();

    std::vector<DropItem> getDrop() override;

    void gainExp(float newExp);
```

jul 21, 20 15:20 **GameCharacter.h** Page 2/2

```

    bool isItem() override;

    bool canDropsItems() override;

    float getMaxLife() override;

    void cure();

    void update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjec
ts, Board &board) override;

    void receiveDamage(float damage, WeaponID weaponId) override;

    bool isDead() override;

    ~GameCharacter();

    RaceID getRace() const;

    void remove(Board &board) override;

    uint getGoldAmount();

    void setGoldAmount(uint goldAmount);

    CharacterStateID getStateId() override;

    float getExp() const;

    std::string getStringInventory() const;

    InputQueue &getQueueInputs();

    NPCInfo interact(GameObject& character, InputInfo input) override;

    void equipItem(int itemToEquip);

    void unequipItem(int itemToUnequip);

    bool takeItem(ItemsInventoryID anItemId, int amount);

    void dropItem(int index);

    void updateHealthAndMana();

    bool canBeAttacked(int enemyLevel) const override;
};

#endif
```

```
#include <iostream>
#include <utility>
#include <states/StateTranslator.h>
#include "GameCharacter.h"
#include "GameStatsConfig.h"
#include "../common/Random.h"
#include "ItemTranslator.h"

PlayerInfo GameCharacter::getPlayerInfo() {
    return PlayerInfo(id, point, goldAmount, life, mana, textureHashId, direction, GameStatsConfig::getMaxGold(level),
                      GameStatsConfig::getMaxHealth(race, gameClass, level),
                      GameStatsConfig::getMaxMana(race, gameClass, level),
                      exp,GameStatsConfig::getNextLevelLimit(level), level,
                      getStringInventory(), StateTranslator::stateToCharacterState(statePool.getStateId()), interactWeapon);
}

GameCharacter::GameCharacter(uint id, RaceID aRace, GameClassID aClass, std::shared_ptr<Cell> initialCell, Point initialPoint):
        GameObject(id, initialPoint, std::move(initialCell)), race(aRace), gameClass(aClass), statePool(*this), queueInputs(true), inventory() {

    this→life = GameStatsConfig::getMaxHealth(race, gameClass, level);
    this→mana = GameStatsConfig::getMaxMana(race, gameClass, level);
    this→goldAmount = 100;
    this→exp = 0;
    this→direction = Direction::down;
    this→textureHashId = updateTextureHashId(); //Solo deberÃ-a tener la cabeza correspondiente y su cuerpo. "ht00|h03|b01|s00|w00"
}

void GameCharacter::update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {
    this→infoInteracting.type = 0;
    statePool.updateState();
    statePool.performTask(gameObjects, board);
    this→textureHashId = updateTextureHashId();
    updateHealthAndMana();
}

std::string GameCharacter::updateTextureHashId() {
    std::string equipment;
    std::string idHelmet = std::to_string((int)this→helmet);
    equipment += "ht";
    if (idHelmet.size() ≡ 1)
        equipment += "0";
    equipment += idHelmet + "|";
    equipment += "h";
    if (¬isDead()){
        std::string idHead = std::to_string((int)this→race);
        if (idHead.size() ≡ 1)
            equipment += "0";
        equipment += idHead + "|";
    } else {
        equipment += "00|";
    }
    equipment += "b";
    if (this→body ≡ BodyID::Nothing ∨ (this→body ≡ BodyID::Ghost ∧ ¬isDead())) {
        this→body = (BodyID)(Random::get(1,3));
    }
    std::string idBody = std::to_string((int)this→body);
    if (idBody.size() ≡ 1)
        equipment += "0";
    equipment += idBody + "|";
```



```
    std::string idShield = std::to_string((int)this→shield);
    equipment += "s";
    if (idShield.size() ≡ 1)
        equipment += "0";
    equipment += idShield + "|";
    equipment += "w";
    std::string idWeapon = std::to_string((int)this→weapon);
    if (idWeapon.size() ≡ 1)
        equipment += "0";
    equipment += idWeapon;
    return equipment;
}

std::string GameCharacter::getStringInventory() const {
    return inventory.getStringInventory();
}

void GameCharacter::equipItem(int itemToEquip) {
    ItemsInventoryID idItem = inventory.getItem(itemToEquip-1);
    if (idItem ≡ ItemsInventoryID::Nothing)
        return;
    ItemInfo info = GameStatsConfig::getItem(idItem);
    ItemsInventoryID item = ItemsInventoryID::Nothing;
    if (info.type ≡ "Weapon") {
        WeaponID newWeapon = ItemTranslator::itemToWeapon(idItem);
        item = ItemTranslator::weaponToItem(this→weapon);
        this→weapon = newWeapon;
    } else if (info.type ≡ "Body") {
        BodyID newBody = ItemTranslator::itemToBody(idItem);
        item = ItemTranslator::bodyToItem(this→body);
        this→body = newBody;
    } else if(info.type ≡ "Shield") {
        ShieldID newShield = ItemTranslator::itemToShield(idItem);
        item = ItemTranslator::shieldToItem(this→shield);
        this→shield = newShield;
    } else if(info.type ≡ "Helmet") {
        HelmetID newHelmet = ItemTranslator::itemToHelmet(idItem);
        item = ItemTranslator::helmetToItem(this→helmet);
        this→helmet = newHelmet;
    } else if(info.type ≡ "Potion") {
        this→consumePotion(info);
    }
    this→inventory.removeItem(idItem);
    this→inventory.addItem(item);
}

void GameCharacter::unequipItem(int itemToUnequip) {
    ItemsInventoryID item = ItemsInventoryID::Nothing;
    if (¬inventoryIsFull()) {
        switch (itemToUnequip) {
            case 0:
                item = ItemTranslator::helmetToItem(this→helmet);
                this→helmet = HelmetID::Nothing;
                break;
            case 1:
                item = ItemTranslator::weaponToItem(this→weapon);
                this→weapon = WeaponID::Nothing;
                break;
            case 2:
                item = ItemTranslator::shieldToItem(this→shield);
                this→shield = ShieldID::Nothing;
                break;
        }
    }
    this→inventory.removeItem(ItemsInventoryID::Nothing);
    this→inventory.addItem(item);
```

jul 21, 20 15:20 **GameCharacter.cpp** Page 3/6

```
126 }
127
128 void GameCharacter::consumePotion(const ItemInfo& potion) {
129     uint maxMana = GameStatsConfig::getMaxMana(this→race, this→gameClass,this→
    level);
130     uint maxHealth = GameStatsConfig::getMaxHealth(this→race, this→gameClass,t
    his→level);
131     mana = mana + potion.manaRestored > maxMana ? maxMana : mana + potion.manaRe
    stored;
132     life = life + potion.healthRestored > maxHealth ? maxHealth : life + potion.
    healthRestored;
133 }
134
135 RaceID GameCharacter::getRace() const {
136     return race;
137 }
138
139 uint GameCharacter::getGoldAmount() {
140     return goldAmount;
141 }
142
143 float GameCharacter::getExp() const {
144     return exp;
145 }
146
147 InputQueue &GameCharacter::getQueueInputs() {
148     return queueInputs;
149 }
150 CharacterStateID GameCharacter::getStateId() {
151     return StateTranslator::stateToCharacterState(statePool.getStateId());
152 }
153
154 void GameCharacter::receiveDamage(float damage, WeaponID weaponId) {
155     setInteractWeapon(weaponId);
156     if (GameStatsConfig::canEvade(race)) {
157         std::cout << "Enemy fail attack" << std::endl;
158     } else {
159         float defense = GameStatsConfig::getDefense(body, shield, helmet);
160         float realDamage = damage − defense;
161         if (realDamage > 0) {
162             life = (life − realDamage > 0) ? life − realDamage : 0;
163         }
164         std::cout << "Enemy attack damage: " << damage << std::endl;
165         std::cout << "Character defense: " << defense << std::endl;
166         std::cout << "Enemy real damage: " << realDamage << std::endl;
167     }
168     if (isDead()) {
169         this→mana = 0;
170         body = BodyID::Ghost;
171         shield = ShieldID::Nothing;
172         weapon = WeaponID::Nothing;
173         helmet = HelmetID::Nothing;
174         exp = exp − GameStatsConfig::getLoseExp() < 0 ? 0 : exp − GameStatsConfi
    g::getLoseExp();
175     }
176 }
177
178 bool GameCharacter::isDead() {
179     return life ≡ 0;
180 }
181
182 bool GameCharacter::hasAnInputInfo() {
183     return ¬queueInputs.empty();
184 }
185
186 InputInfo GameCharacter::getNextInputInfo() {
```

jul 21, 20 15:20 **GameCharacter.cpp** Page 4/6

```
187     return queueInputs.pop();;
188 }
189
190 WeaponID GameCharacter::getWeapon() {
191     return weapon;
192 }
193
194 void GameCharacter::cure(){
195     this→life = GameStatsConfig::getMaxHealth(race,gameClass, level);
196     this→mana = GameStatsConfig::getMaxMana(race,gameClass, level);
197
198 }
199
200 NPCInfo GameCharacter::interact(GameObject& character, InputInfo input) {
201     NPCInfo info;
202     return info;
203 }
204
205 bool GameCharacter::isReadyToRemove() {
206     return false;
207 }
208
209 void GameCharacter::remove(Board &board) {
210     cell→free();
211 }
212
213 void GameCharacter::gainExp(float newExp) {
214     exp += newExp;
215 }
216
217 float GameCharacter::getMaxLife() {
218     return GameStatsConfig::getMaxHealth(race, gameClass, level);
219 }
220
221 bool GameCharacter::inventoryIsFull() {
222     return inventory.isFull();
223 }
224
225 bool GameCharacter::addItemToInventory(ItemsInventoryID aItemInventoryId) {
226    return inventory.addItem(aItemInventoryId);
227 }
228
229 void GameCharacter::setGoldAmount(uint aGoldAmount) {
230     GameCharacter::goldAmount = aGoldAmount;
231 }
232
233 ItemsInventoryID GameCharacter::removeItemFromInventory(ItemsInventoryID aItemTo
    Remove) {
234     return inventory.removeItem(aItemToRemove);
235 }
236
237 void GameCharacter::gainGold(int aGoldAmount) {
238     goldAmount += aGoldAmount;
239 }
240
241 std::vector<DropItem> GameCharacter::getDrop() {
242     std::vector<DropItem> dropsItems;
243     if (isDead()) {
244         for (auto &aInventoryItem : inventory.getInventoryItems()) {
245             if (aInventoryItem ≠ ItemsInventoryID::Nothing) {
246                 dropsItems.emplace_back(aInventoryItem, 1);
247             }
248         }
249         int diffGold = goldAmount − GameStatsConfig::getMaxGold(level);
250         if (diffGold > 0) {
251             goldAmount = GameStatsConfig::getMaxGold(level);
```

```
252             dropsItems.emplace_back(ItemsInventoryID::Gold, diffGold);
253         }
254         inventory.clear();
255     } else {
256         dropsItems.emplace_back(itemToDrop, 1);
257         itemToDrop = ItemsInventoryID::Nothing;
258     }
259     return dropsItems;
260 }
261
262 bool GameCharacter::isItem() {
263     return false;
264 }
265
266 bool GameCharacter::canDropsItems() {
267     return (isDead() ∧ ¬inventory.isEmpty()) ∨ itemToDrop ≠ ItemsInventoryID::N
    othing;
268 }
269
270 void GameCharacter::consumeMana() {
271     if (weapon ≠ WeaponID::Nothing) {
272         mana −= GameStatsConfig::getWeaponCost(weapon);
273     }
274 }
275
276 void GameCharacter::upLevel() {
277     level++;
278     life = getMaxLife();
279     mana = GameStatsConfig::getMaxMana(race, gameClass, level);
280 }
281
282 bool GameCharacter::canUseWeapon() {
283     return weapon ≠ WeaponID::Nothing ∧ GameStatsConfig::getWeaponCost(weapon) ≤
     mana;
284 }
285
286 void GameCharacter::updateHealthAndMana() {
287     if (¬isDead()) {
288         float manaMax = GameStatsConfig::getMaxMana(race, gameClass, level);
289         float lifeIncrement = GameStatsConfig::getRecoveryHealth(race);
290         life = life + lifeIncrement > getMaxLife() ? getMaxLife() : lifeIncremen
    t + life;
291         float manaIncrement = statePool.isMeditating() ?
292                               GameStatsConfig::getRecoveryManaMeditation(race, g
    ameClass) :
293                               GameStatsConfig::getRecoveryMana(race);
294         mana = mana + manaIncrement > manaMax ? manaMax : mana + manaIncrement;
295     }
296 }
297
298 bool GameCharacter::takeItem(ItemsInventoryID anItemId, int amount) {
299     bool canTake;
300     if (anItemId ≡ ItemsInventoryID::Gold) {
301         goldAmount += amount;
302         canTake = true;
303     } else {
304         canTake = inventory.addItem(anItemId);
305     }
306     return canTake;
307 }
308
309 void GameCharacter::dropItem(int index) {
310     ItemsInventoryID itemToRemove = inventory.getItem(index);
311     if (itemToRemove ≠ ItemsInventoryID::Nothing) {
312         itemToDrop =  this→inventory.removeItem(itemToRemove);
313     }
```



```
314 }
315
316 bool GameCharacter::canBeAttacked(int enemyLevel) const {
317     return GameStatsConfig::canAttack(this→level, enemyLevel);
318 }
319
320 bool GameCharacter::restoreHealth() {
321     bool canRestoreHealth = false;
322     float healthRestore = 0;
323     if (canUseWeapon()) {
324         setInteractWeapon(weapon);
325         healthRestore = GameStatsConfig::restoreHealth(weapon);
326         canRestoreHealth = true;
327         consumeMana();
328         life = healthRestore + life > getMaxLife() ? getMaxLife() : healthRestor
    e + life;
329     }
330     return canRestoreHealth;
331 }
332
333 GameCharacter::~GameCharacter()= default;
```

jul 21, 20 15:20 **DropItem.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_DROPITEM_H
#define ARGENTUM_TALLER_DROPITEM_H

#include "../common/Identificators.h"

class DropItem {
private:
    ItemsInventoryID id;
    int amount{};

public:
    DropItem(ItemsInventoryID id, int amount);

    ItemsInventoryID getId() const;

    int getAmount() const;

    virtual ~DropItem();
};


#endif //ARGENTUM_TALLER_DROPITEM_H
```

jul 21, 20 15:20 **DropItem.cpp** Page 1/1

```cpp
#include "DropItem.h"

DropItem::DropItem(ItemsInventoryID id, int amount) : id(id), amount(amount) {}

ItemsInventoryID DropItem::getId() const {
    return id;
}

int DropItem::getAmount() const {
    return amount;
}

DropItem::~DropItem() = default;
```

jul 21, 20 15:20 **Creature.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_CREATURE_H
#define ARGENTUM_TALLER_CREATURE_H

#include <states/StatePoolCreature.h>
#include "GameObject.h"
#include "states/State.h"

class Creature : public GameObject {
private:
    uint life{};
    CreatureID creatureId;
    std::unique_ptr<State> state;
    bool itemDrop = false;
    uint8_t timeToRemove = 10;
    StatePoolCreature statePool;

    InputInfo generateRandomInputInfo();
public:
    Creature(uint id, CreatureID creatureId, std::shared_ptr<Cell> initialCell, Point initialPoint);

    void update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) override;

    void notify(uint pursuitId);

    CreatureID getCreatureId() const;

    PlayerInfo getPlayerInfo() override;

    bool hasAnInputInfo() override;

    InputInfo getNextInputInfo() override;

    bool isDead();

    bool canDropsItems() override;

    bool isItem() override;

    std::vector<DropItem> getDrop() override;

    float getMaxLife() override;

    void receiveDamage(float damage, WeaponID weaponId) override;

    CharacterStateID getStateId() override;

    virtual NPCInfo interact(GameObject& character, InputInfo input);

    virtual ~Creature();

    void remove(Board &board) override;

    bool isReadyToRemove() override;

    bool canBeAttacked(int enemyLevel) const;

};


#endif //ARGENTUM_TALLER_CREATURE_H
```

jul 21, 20 15:20 **Creature.cpp** Page 1/3

```cpp
#include <iostream>
#include <utility>
#include <states/StateTranslator.h>
#include "Creature.h"
#include "../common/Random.h"

void Creature::update(std::unordered_map<uint, std::shared_ptr<GameObject>> &gameObjects, Board &board) {
    statePool.updateState();
    statePool.performTask(gameObjects, board);
}

Creature::Creature(uint id, CreatureID creatureId, std::shared_ptr<Cell> initialCell, Point initialPoint) :
        GameObject(id, initialPoint, std::move(initialCell)), creatureId(creatureId), statePool(*this) {

    switch (creatureId) {
        case CreatureID::Goblin:
            this→textureHashId = "ht00|h00|b11|s00|w00";
            break;
        case CreatureID::Skeleton:
            this→textureHashId = "ht00|h00|b12|s00|w00";
            break;
        case CreatureID::Spider:
            this→textureHashId = "ht00|h00|b13|s00|w00";
            break;
        case CreatureID::Zombie:
            this→textureHashId = "ht00|h06|b14|s00|w00";
            break;
        default:
            this→textureHashId = nullptr;
            break;
    }
    life = GameStatsConfig::getMaxHealth(creatureId, level);
}

void Creature::notify(uint pursuitId) {
    if (statePool.startChasing(pursuitId)) {
        InputInfo aInputInfo;
        aInputInfo.aditional = pursuitId;
        statePool.setNextState(StateID::Pursuit, aInputInfo);
    }
}

InputInfo Creature::generateRandomInputInfo() {
    uint8_t randomId = Random::get(2,5);
    InputInfo inputInfo;
    inputInfo.input = InputID(randomId);
    return inputInfo;
}

CharacterStateID Creature::getStateId() {
    return StateTranslator::stateToCharacterState(statePool.getStateId());
}

CreatureID Creature::getCreatureId() const {
    return creatureId;
}

void Creature::receiveDamage(float damage, WeaponID weaponId) {
    setInteractWeapon(weaponId);
    if (GameStatsConfig::canEvade(creatureId)) {
        std::cout << "Enemy fail attack" << std::endl;
    } else {
        float defense = GameStatsConfig::getDefense(creatureId);
```

```
64          float realDamage = damage − defense;
65          if (realDamage > 0) {
66              life = (life − realDamage > 0) ? life − realDamage : 0;
67          }
68          std::cout << "Character attack damage: " << damage << std::endl;
69          std::cout << "Enemy defense: " << defense << std::endl;
70          std::cout << "Character real damage: " << realDamage << std::endl;
71          std::cout << "Enemy life is: " << life << std::endl;
72          if (isDead()) {
73              itemDrop = true;
74              std::cout << "Enemy is dead" << realDamage << std::endl;
75          }
76      }
77  }
78
79  bool Creature::isDead() {
80      return life ≡ 0;
81  }
82
83  NPCInfo Creature::interact(GameObject& character, InputInfo input) {
84      NPCInfo info;
85      return info;
86  }
87
88  bool Creature::isReadyToRemove() {
89      if (isDead()) {
90          timeToRemove−−;
91      }
92      return timeToRemove ≡ 0;
93  }
94
95  void Creature::remove(Board &board) {
96      board.removeCreatureFromNest(cell);
97      cell→free();
98  }
99
100 float Creature::getMaxLife() {
101     return GameStatsConfig::getMaxHealth(creatureId, level);
102 }
103
104 std::vector<DropItem> Creature::getDrop() {
105     std::vector<DropItem> dropItems;
106     int randomDrop = Random::get(0,100);
107     ItemsInventoryID aItemInventoryId = ItemsInventoryID::Nothing;
108     float amount = 1;
109     if ((randomDrop > 80) ∧ randomDrop ≤ 96) {
110         aItemInventoryId = ItemsInventoryID::Gold;
111         amount = GameStatsConfig::getGoldDrop(creatureId, level);
112     } else if (randomDrop > 96 ∧ randomDrop ≤ 98) {
113         aItemInventoryId = ItemsInventoryID(Random::get(21, 22));
114     } else if (randomDrop > 98 ∧ randomDrop ≤ 100) {
115         aItemInventoryId = ItemsInventoryID(Random::get(1, 20));
116     }
117     if (aItemInventoryId ≠ ItemsInventoryID::Nothing) {
118         dropItems.emplace_back(aItemInventoryId, amount);
119     }
120     itemDrop = false;
121     return dropItems;
122 }
123
124 bool Creature::isItem() {
125     return false;
126 }
127
128 bool Creature::canDropsItems() {
129     return itemDrop;
```



```
130 }
131
132 bool Creature::canBeAttacked(int enemyLevel) const {
133     return true;
134 }
135
136 PlayerInfo Creature::getPlayerInfo() {
137     return GameObject::getPlayerInfo();
138 }
139
140 bool Creature::hasAnInputInfo() {
141     return true;
142 }
143
144 InputInfo Creature::getNextInputInfo() {
145     return generateRandomInputInfo();
146 }
147
148
149 Creature::~Creature() = default;
```

jul 21, 20 15:20 **Cell.h** Page 1/1

```
#ifndef ARGENTUM_TALLER_CELL_H
#define ARGENTUM_TALLER_CELL_H

#include <zconf.h>

class Cell {
private:
    uint x;
    uint y;
    bool empty;
    uint gameObjectId;
    bool city;
    uint nestId;
    uint itemId;
    bool priest;
public:
    Cell(uint x, uint y);

    void occupied(uint id);

    void free();

    std::tuple<int, int> getCoord();

    friend bool operator≡(const Cell& c1, const Cell& c2);

    friend bool operator≠(const Cell& c1, const Cell& c2);

    void removeItem();

    bool hasItem() const;

    void setItemId(uint itemId);

    uint getItemId() const;

    bool isEmpty() const;

    bool isPriest() const;

    void addPriest();

    void setEmpty(bool empty);

    uint getGameObjectId() const;

    bool isCity() const;

    void setCity(bool city);

    uint getNestId() const;

    void setNestId(uint nestId);

    uint getX() const;

    uint getY() const;

    virtual ~Cell();

};


#endif //ARGENTUM_TALLER_CELL_H
```

jul 21, 20 15:20 **Cell.cpp** Page 1/2

```
#include <tuple>
#include "Cell.h"

Cell::Cell(uint x, uint y) : x(x), y(y), empty(true), gameObjectId(0), city(false), nestId(0), itemId(0), priest(false) {}

bool Cell::isEmpty() const {
    return empty;
}

void Cell::setEmpty(bool empty) {
    Cell::empty = empty;
}

uint Cell::getGameObjectId() const {
    return gameObjectId;
}

bool Cell::isCity() const {
    return city;
}

void Cell::setCity(bool city) {
    Cell::city = city;
}

uint Cell::getNestId() const {
    return nestId;
}

void Cell::setNestId(uint nestId) {
    Cell::nestId = nestId;
}

uint Cell::getX() const {
    return x;
}

uint Cell::getY() const {
    return y;
}

void Cell::occupied(uint id) {
    gameObjectId = id;
    empty = false;
}

void Cell::free() {
    gameObjectId = 0;
    empty = true;
}

std::tuple<int, int> Cell::getCoord() {
    return {x, y};
}

bool operator≡(const Cell& c1, const Cell &c2) {
    return c1.x ≡ c2.x ∧ c1.y ≡ c2.y;
}

bool operator≠(const Cell& c1, const Cell &c2) {
    return ¬(c1≡c2);
}

void Cell::removeItem() {
    itemId = 0;
```

jul 21, 20 15:20 **Cell.cpp** Page 2/2

```
66  }
67
68  uint Cell::getItemId() const {
69      return itemId;
70  }
71
72  bool Cell::hasItem() const {
73      return itemId ≠ 0;
74  }
75
76  void Cell::setItemId(uint itemId) {
77      Cell::itemId = itemId;
78  }
79
80  bool Cell::isPriest() const {
81      return priest;
82  }
83
84  void Cell::addPriest() {
85      priest = true;
86  }
87
88
89  Cell::~Cell() = default;
```

jul 21, 20 15:20 **Board.h** Page 1/2

```
1   #ifndef ARGENTUM_TALLER_BOARD_H
2   #define ARGENTUM_TALLER_BOARD_H
3
4
5   #include <vector>
6   #include <memory>
7   #include <queue>
8   #include "../common/Point.h"
9   #include "../common/StaticObject.h"
10  #include "../common/ObjectLayer.h"
11  #include "../common/PlayerInfo.h"
12  #include "GameStats.h"
13  #include "GameStatsConfig.h"
14  #include "Nest.h"
15  #include "NestContainer.h"
16  #include "Cell.h"
17  #include "../common/TiledMap.h"
18
19  class Board {
20  private:
21      uint width{};
22      uint height{};
23      uint cols{}, rows{};
24      Point initialPoint;
25      NestContainer nestContainer;
26      std::vector<std::vector<std::shared_ptr<Cell>>> cells;
27
28      void addCity(StaticObject &city);
29      std::tuple<int, int> convertPoint(const Point &point);
30      void addCollisonObject(StaticObject &object);
31      std::pair<int, int> getCorrectPosition(const std::shared_ptr<Cell>& aCell, D
    irection aDirection);
32  public:
33      Board();
34
35      Board(TiledMap &map, uint8_t nestCreaturesLimit);
36
37      virtual ~Board();
38
39      std::shared_ptr<Cell> getCellFromPoint(const Point& aPoint);
40
41      Point getPointFromCell(const std::shared_ptr<Cell>& aCell);
42
43      std::shared_ptr<Cell> getCell(uint x, uint y);
44
45      std::shared_ptr<Cell> getInitialCell();
46
47      std::vector<std::pair<uint8_t, std::shared_ptr<Cell>>> getAdjacents(std::tup
    le<uint, uint>, uint8_t distance);
48
49      uint8_t getDistance(const std::shared_ptr<Cell>& firstCell, const std::share
    d_ptr<Cell>& secondCell);
50
51      std::shared_ptr<Cell> getInitialCellInNest(Nest& nest);
52
53      std::vector<uint> getCreaturesInNest(uint nestId);
54
55      Nest& getAvailableNest();
56
57      std::vector<std::shared_ptr<Cell>> setCellsInNest(const std::shared_ptr<Cell
    >& aNestCell, uint nestId);
58
59      bool characterCanMove(const std::shared_ptr<Cell>& aCell, Direction aDirecti
    on);
60
61      bool creatureCanMove(const std::shared_ptr<Cell>& aCell, Direction aDirectio
```

jul 21, 20 15:20 **Board.h** Page 2/2

```
    n);
62
63      std::shared_ptr<Cell> getNextCell(const std::shared_ptr<Cell>& aCell, Direct
    ion aDirection);
64
65      std::shared_ptr<Cell> getBestCell(const std::shared_ptr<Cell>& actualCell, c
    onst std::shared_ptr<Cell>& DestinationCell, bool overstepNest);
66
67      Direction getDirection(const std::shared_ptr<Cell>& actualCell, const std::s
    hared_ptr<Cell>& DestinationCell);
68
69      void removeCreatureFromNest(const std::shared_ptr<Cell>& aCell);
70
71      int getAmountCreatures();
72
73      std::shared_ptr<Cell> getNextEmptyCell(const std::shared_ptr<Cell>& aCell);
74
75      std::shared_ptr<Cell> getCloserPriest(const std::shared_ptr<Cell>& aCell);
76
77  };
78
79
80  #endif //ARGENTUM_TALLER_BOARD_H
```

jul 21, 20 15:20 **Board.cpp** Page 1/5

```
1   #include "Board.h"
2   #include "Node.h"
3   #include "NodeContainer.h"
4   #include <memory>
5   #include <cstdlib>
6
7   Board::~Board() = default;
8
9   Board::Board() = default;
10
11  Board::Board(TiledMap &map, uint8_t nestCreaturesLimit) : initialPoint(1125, 550
    ) {
12      cols = map.getWidth();
13      rows = map.getHeight();
14      height = map.getHeight() * map.getTileHeight();
15      width = map.getWidth() * map.getTileWidth();
16      uint nestIdCounter = 1;
17      for (size_t i = 0; i < rows; ++i) {
18          std::vector<std::shared_ptr<Cell>> cellsRows;
19          for (size_t j = 0; j < cols; ++j) {
20              cellsRows.push_back(std::make_shared<Cell>(i, j));
21          }
22          cells.push_back(cellsRows);
23      }
24      for (auto∧ anObjectLayer : map.getObjectLayers()) {
25          if (anObjectLayer.getName() ≡ "cities") {
26              for (StaticObject& aCity : anObjectLayer.getObjects()) {
27                  addCity(aCity);
28              }
29          } else if (anObjectLayer.getName() ≡ "collisionObjects") {
30              for (StaticObject &aCollisionObject : anObjectLayer.getObjects()) {
31                  addCollisonObject(aCollisionObject);
32              }
33          } else if (anObjectLayer.getName() ≡ "nestPoints") {
34              std::vector<Nest> nests;
35              std::shared_ptr<Cell> aCell;
36              uint nestId;
37              for (StaticObject& aNest : anObjectLayer.getObjects()) {
38                  nestId = nestIdCounter++;
39                  aCell = getCellFromPoint(aNest.getTopLeft());
40                  nests.emplace_back(nestCreaturesLimit, nestId, setCellsInNest(aC
    ell, nestId));
41              }
42              nestContainer = NestContainer(nests);
43          }
44      }
45  }
46
47  Nest& Board::getAvailableNest() {
48      return nestContainer.getNextNestAvailable();
49  }
50
51  std::vector<uint> Board::getCreaturesInNest(uint nestId) {
52      Nest& nest = nestContainer.getNest(nestId);
53      return nest.getCreatures();
54  }
55
56  void Board::addCity(StaticObject &city) {
57      std::tuple<int, int> topLeft = convertPoint(city.getTopLeft());
58      std::tuple<int, int> bottomRight = convertPoint(city.getBottomRight());
59      for (int i = std::get<0>(topLeft); i < std::get<0>(bottomRight); ++i) {
60          for (int j = std::get<1>(topLeft); j < std::get<1>(bottomRight); ++j) {
61              cells[i][j]→setCity(true);
62          }
63      }
64  }
```

```
65
66  std::tuple<int, int> Board::convertPoint(const Point &point) {
67      int x = point.x / (width / cols);
68      int y  = point.y / (height / rows);
69      return std::make_tuple(x, y);
70  }
71
72  void Board::addCollisonObject(StaticObject &collisionObject) {
73      std::tuple<int, int> topLeft = convertPoint(collisionObject.getTopLeft());
74      std::tuple<int, int> bottomRight = convertPoint(collisionObject.getBottomRig
    ht());
75      int diffX = std::get<0>(bottomRight) − std::get<0>(topLeft);
76      int diffY = std::get<1>(bottomRight) − std::get<1>(topLeft);
77      for (int i = std::get<0>(topLeft); diffX > 0 ? i < std::get<0>(bottomRight)
    : i ≤ std::get<0>(bottomRight); ++i) {
78          for (int j = std::get<1>(topLeft); diffY > 0 ? j < std::get<1>(bottomRig
    ht) : j ≤ std::get<1>(bottomRight); ++j) {
79              cells[i][j]→setEmpty(false);
80          }
81      }
82  }
83
84  std::shared_ptr<Cell> Board::getCellFromPoint(const Point &aPoint) {
85      std::tuple<uint, uint> position = convertPoint(aPoint);
86      return cells[std::get<0>(position)][std::get<1>(position)];
87  }
88
89  std::shared_ptr<Cell> Board::getCell(uint x, uint y) {
90      return cells[x][y];
91  }
92
93  std::shared_ptr<Cell> Board::getInitialCell() {
94      std::shared_ptr<Cell> cell = getCellFromPoint(initialPoint);
95      if (→cell→isEmpty()) {
96          for (auto &adjacent : getAdjacents(convertPoint(initialPoint), 4)) {
97              if (adjacent.second→isEmpty()) {
98                  cell = adjacent.second;
99                  break;
100             }
101         }
102     }
103     return cell;
104 }
105
106 std::vector<std::pair<uint8_t, std::shared_ptr<Cell>>> Board::getAdjacents(std::
    tuple<uint, uint> position, uint8_t distance) {
107     std::vector<std::pair<uint8_t, std::shared_ptr<Cell>>> adjacents;
108     uint x = std::get<0>(position);
109     uint y = std::get<1>(position);
110     std::shared_ptr<Cell> originCell = getCell(x, y);
111     uint leftLimitX = x − distance ≥ 0 ? x − distance : 0;
112     uint rightLimitX = x + distance < cols ? x + distance : cols − 1;
113     uint leftLimitY = y − distance ≥ 0 ? y − distance : 0;
114     uint rightLimitY = y + distance < rows ? y + distance : rows − 1;
115     std::shared_ptr<Cell> aCell;
116     for (size_t i = leftLimitX; i ≤ rightLimitX; ++i) {
117         for (size_t j = leftLimitY; j ≤ rightLimitY; ++j) {
118             if (i ≡ x ∧ j ≡ y) {
119                 continue;
120             }
121             aCell = getCell(i, j);
122             uint8_t aDistance = getDistance(originCell, aCell);
123             if (aDistance ≤ distance) {
124                 adjacents.emplace_back(aDistance, aCell);
125             }
126         }
```



```
127     }
128     std::sort_heap(adjacents.begin(), adjacents.end());
129     return adjacents;
130 }
131
132 uint8_t Board::getDistance(const std::shared_ptr<Cell>& firstCell, const std::sh
    ared_ptr<Cell>& secondCell) {
133     return std::abs(int(firstCell→getX() − secondCell→getX())) + std::abs(int(
    firstCell→getY() − secondCell→getY()));
134 }
135
136 Point Board::getPointFromCell(const std::shared_ptr<Cell>& aCell) {
137     return Point(aCell→getX() * (width/cols), aCell→getY() * (height/rows));
138 }
139
140 std::shared_ptr<Cell> Board::getInitialCellInNest(Nest &nest) {
141     return nest.getFreeCell();
142 }
143
144 std::vector<std::shared_ptr<Cell>> Board::setCellsInNest(const std::shared_ptr<C
    ell>& aNestCell, uint nestId) {
145     std::vector<std::shared_ptr<Cell>> cellInsideNest;
146     uint distance = 8;
147     uint leftLimitX = int(aNestCell→getX() − distance) ≥ 0 ? aNestCell→getX()
    − distance : 0;
148     uint rightLimitX = aNestCell→getX() + distance < cols ? aNestCell→getX() +
     distance : cols − 1;
149     uint leftLimitY = int(aNestCell→getY() − distance) ≥ 0 ? aNestCell→getY()
    − distance : 0;
150     uint rightLimitY = aNestCell→getY() + distance < rows ? aNestCell→getY() +
     distance : rows − 1;
151     std::shared_ptr<Cell> aCell;
152     for (size_t i = leftLimitY; i ≤ rightLimitY; ++i) {
153         for (size_t j = leftLimitX; j ≤ rightLimitX; ++j) {
154             aCell = getCell(i, j);
155             if (→aCell→isCity()) {
156                 aCell→setNestId(nestId);
157                 cellInsideNest.push_back(aCell);
158             }
159         }
160     }
161     return cellInsideNest;
162 }
163
164 bool Board::characterCanMove(const std::shared_ptr<Cell> &aCell, Direction aDire
    ction) {
165     bool canMove = false;
166     std::pair<uint, uint> aPosition = getCorrectPosition(aCell, aDirection);
167     if (aPosition.first ≥ 0 ∧ aPosition.first < cols ∧ aPosition.second ≥ 0 ∧ aP
    osition.second < rows) {
168         canMove = getCell(aPosition.first, aPosition.second)→isEmpty();
169     }
170     return canMove;
171 }
172
173 std::shared_ptr<Cell> Board::getNextCell(const std::shared_ptr<Cell> &aCell, Dir
    ection aDirection) {
174     std::pair<uint, uint> aPosition = getCorrectPosition(aCell, aDirection);
175     return getCell(aPosition.first, aPosition.second);
176 }
177
178 bool Board::creatureCanMove(const std::shared_ptr<Cell> &aCell, Direction aDirec
    tion) {
179     bool canMove = false;
180     std::pair<uint, uint> aPosition = getCorrectPosition(aCell, aDirection);
181     if (aPosition.first ≥ 0 ∧ aPosition.first < cols ∧ aPosition.second ≥ 0 ∧ aP
```

```
    osition.second < rows) {
182         std::shared_ptr<Cell> nestCell = getCell(aPosition.first, aPosition.seco
    nd);
183         canMove = nestCell→isEmpty() ∧ nestCell→getNestId() ≡ aCell→getNestId(
    );
184     }
185     return canMove;
186 }
187
188 std::pair<int, int> Board::getCorrectPosition(const std::shared_ptr<Cell>& aCell
    , Direction aDirection) {
189     int x = aCell→getX();
190     int y = aCell→getY();
191     switch(aDirection) {
192         case Direction::up:
193             y −−;
194             break;
195         case Direction::down:
196             y ++;
197             break;
198         case Direction::left:
199             x −−;
200             break;
201         case Direction::right:
202             x ++;
203             break;
204     }
205     return {x, y};
206 }
207
208 std::shared_ptr<Cell> Board::getBestCell(const std::shared_ptr<Cell>& actualCell
    , const std::shared_ptr<Cell>& destinationCell, bool overstepNest) {
209     NodeContainer close;
210     NodeContainer open;
211     Node firstNode = Node(0, 0, getDistance(actualCell, destinationCell), actual
    Cell);
212     Node destinationNode = Node(0, 0, 0, destinationCell);
213     Node& pos = firstNode;
214     open.insert(firstNode);
215     while (¬open.has(destinationNode)) {
216         pos = open.getBestNode();
217         for (auto &adjacent : getAdjacents(pos.getCell()→getCoord(), 1)) {
218             Node aNode = Node(pos.getId(), pos.getG() + 1, getDistance(adjacent.
    second, destinationCell), adjacent.second);
219             if (overstepNest ∨ adjacent.second→getNestId() ≡ actualCell→getNes
    tId()) {
220                 if (adjacent.second ≡ destinationCell) {
221                     open.insert(aNode);
222                 } else if (adjacent.second→isEmpty() ∧ ¬close.has(aNode)) {
223                     if (open.has(aNode)) {
224                         open.modifyNode(aNode.getId(), pos);
225                     } else {
226                         open.insert(aNode);
227                     }
228                 }
229             }
230         }
231         close.insert(pos);
232     }
233     Node &selectedNode = close.get(open.get(destinationNode.getId()).getParent()
    );
234     while (close.get(selectedNode.getParent()).getParent() ≠ 0) {
235         selectedNode = close.get(selectedNode.getParent());
236     }
237     return selectedNode.getCell();
238 }
```



```
239
240 Direction Board::getDirection(const std::shared_ptr<Cell>& actualCell, const std
    ::shared_ptr<Cell>& destinationCell) {
241     Direction aDirection;
242     if (actualCell→getX() > destinationCell→getX()) {
243         aDirection = Direction::left;
244     }
245     if (actualCell→getX() < destinationCell→getX()) {
246         aDirection = Direction::right;
247     }
248     if (actualCell→getY() > destinationCell→getY()) {
249         aDirection = Direction::up;
250     }
251     if (actualCell→getY() < destinationCell→getY()) {
252         aDirection = Direction::down;
253     }
254     return aDirection;
255 }
256
257 void Board::removeCreatureFromNest(const std::shared_ptr<Cell>& aCell) {
258     Nest &nest = nestContainer.getNest(aCell→getNestId());
259     nest.removeCreature(aCell→getGameObjectId());
260 }
261
262 int Board::getAmountCreatures() {
263     return nestContainer.getAmountCreatures();
264 }
265
266 std::shared_ptr<Cell> Board::getNextEmptyCell(const std::shared_ptr<Cell> &aCell
    ) {
267     std::shared_ptr<Cell> cell = aCell;
268     uint distance = 0;
269     while (¬cell→isEmpty() ∨ cell→hasItem()) {
270         distance++;
271         for (auto &adjacent : getAdjacents(aCell→getCoord(), distance)) {
272             if (adjacent.second→isEmpty() ∧ ¬adjacent.second→hasItem()) {
273                 cell = adjacent.second;
274                 break;
275             }
276         }
277     }
278     return cell;
279 }
280
281 std::shared_ptr<Cell> Board::getCloserPriest(const std::shared_ptr<Cell>& aCell)
     {
282     std::shared_ptr<Cell> chosenCell;
283     std::shared_ptr<Cell> aCellPriest;
284     uint8_t minDistance = 0;
285     for (size_t i = 0; i < rows; ++i) {
286         for (size_t j = 0; j < cols; ++j) {
287             aCellPriest = getCell(i, j);
288             if (aCellPriest→isPriest()) {
289                 if (minDistance ≡ 0 ∨ minDistance > getDistance(aCellPriest, aCe
    ll)) {
290                     minDistance = getDistance(aCellPriest, aCell);
291                     chosenCell = aCellPriest;
292                 }
293             }
294         }
295     }
296     return chosenCell;
297 }
```

jul 21, 20 15:20 **Banker.h** Page 1/1

```h
#ifndef BANKER_H
#define BANKER_H

#include <unordered_map>
#include <vector>
#include "../common/Identificators.h"
#include "Profession.h"

//Se optÃ³ por realizar esta clase como un Singleton debido a que
//los diversos jugadores(Threads) tendrÃ¡n solo acceso a sus items
//que estÃ©n depositados dentro del Banco, sin poder acceder a ningun
//otra "cuenta" que pertenezca al usuario.
class Banker: public Profession {
private:
    Banker();
    std::unordered_map<uint, std::vector<ItemsInventoryID>> accountsItems;
    std::unordered_map<uint, uint> accountsGold;
    static Banker* banker;

    void createNewAccount(uint accountHolder);

    std::vector<ItemsInventoryID> getMyItems(uint accountHolder);

    uint checkBalance(uint accountHolder);

    ItemsInventoryID retireItem(uint accountHolder, uint item);

    void depositItem(uint accountHolder, ItemsInventoryID idItem);

    void depositGold(uint accountHolder, uint amountGold);

    uint retireGold(uint accountHolder, uint amountGold);

public:
    static Banker* getInstance();

    virtual NPCInfo getInfo(uint id);

    void processInput(GameCharacter &character, InputInfo inputInfo) override;

    virtual ~Banker();
};

#endif
```

jul 21, 20 15:20 **Banker.cpp** Page 1/3

```cpp
#include "Banker.h"

Banker* Banker::banker = nullptr;

Banker::Banker() :accountsItems(), accountsGold() {
    this→actions.push_back(ActionsProfessionID::DepositGold);
    this→actions.push_back(ActionsProfessionID::DepositItem);
    this→actions.push_back(ActionsProfessionID::RetireItem);
    this→actions.push_back(ActionsProfessionID::RetireGold);
}

Banker* Banker::getInstance() {
    if (banker ≡ nullptr)
        banker = new Banker();
    return banker;
}

std::vector<ItemsInventoryID> Banker::getMyItems(uint accountHolder) {
    auto iter = accountsItems.find(accountHolder);
    if (iter ≡ accountsItems.end()) {
        createNewAccount(accountHolder);
        iter = accountsItems.find(accountHolder);
    }
    return (*iter).second;
}

uint Banker::checkBalance(uint accountHolder) {
    auto iter = accountsGold.find(accountHolder);
    if (iter ≡ accountsGold.end()) {
        createNewAccount(accountHolder);
        return 0;
    }
    return accountsGold[accountHolder];
}

void Banker::createNewAccount(uint accountHolder) {
    uint gold = 0;
    accountsGold.insert({accountHolder,gold});
    std::vector<ItemsInventoryID> emptyAccount;
    emptyAccount.push_back(ItemsInventoryID::HealthPotion);
    accountsItems.insert({accountHolder,emptyAccount});
}

ItemsInventoryID Banker::retireItem(uint accountHolder, uint item) {
    auto iter = accountsItems.find(accountHolder);
    if (iter ≡ accountsItems.end()) {
        createNewAccount(accountHolder);
        return ItemsInventoryID::Nothing;
    }
    if (accountsItems[accountHolder].size() ≡ 0)
        return ItemsInventoryID::Nothing;
    ItemsInventoryID itemToRetire = (*iter).second.at(item);
    (*iter).second.at(item) = ItemsInventoryID::Nothing;
    auto it = (*iter).second.begin();
    while (it ≠ (*iter).second.end()) {
        if (*it ≡ ItemsInventoryID::Nothing) {
            it = (*iter).second.erase(it);
        } else {
            it++;
        }
    }
    return itemToRetire;
}

NPCInfo Banker::getInfo(uint id) {
    NPCInfo info;
```

```
67      info.type = 3;
68      info.actions = actions;
69      info.gold = checkBalance(id);
70      info.itemsInBank = getMyItems(id);
71      return info;
72  }
73
74
75  void Banker::depositItem(uint accountHolder, ItemsInventoryID idItem) {
76      auto iter = accountsItems.find(accountHolder);
77      if (iter ≡ accountsItems.end()) {
78          createNewAccount(accountHolder);
79          iter = accountsItems.find(accountHolder);
80      }
81      (*iter).second.push_back(idItem);
82  }
83
84  void Banker::depositGold(uint accountHolder, uint amountGold) {
85      auto iter = accountsGold.find(accountHolder);
86      if (iter ≡ accountsGold.end())
87          createNewAccount(accountHolder);
88      accountsGold[accountHolder] += amountGold;
89  }
90
91  uint Banker::retireGold(uint accountHolder, uint amountGold) {
92      auto iter = accountsGold.find(accountHolder);
93      if (iter ≡ accountsGold.end()) {
94          createNewAccount(accountHolder);
95          return 0;
96      }
97      if (accountsGold[accountHolder] < amountGold) {
98          amountGold = accountsGold[accountHolder];
99          accountsGold[accountHolder] = 0;
100     } else {
101         accountsGold[accountHolder] −= amountGold;
102     }
103     return amountGold;
104 }
105
106 void Banker::processInput(GameCharacter &character, InputInfo inputInfo) {
107     bool addedToInventory;
108     ItemsInventoryID aItem;
109     int gold;
110     switch (inputInfo.input) {
111         case InputID::depositItem:
112             character.removeItemFromInventory(ItemsInventoryID(inputInfo.adition
    al));
113             depositItem(character.getId(),ItemsInventoryID(inputInfo.aditional))
    ;
114             break;
115         case InputID::retireItem:
116             aItem = retireItem(character.getId(), inputInfo.aditional);
117             if (aItem ≠ ItemsInventoryID::Nothing) {
118                 addedToInventory = character.addItemToInventory(aItem);
119                 if(¬addedToInventory)
120                     depositItem(character.getId(), aItem);
121             }
122             break;
123         case InputID::depositGold:
124             gold = character.getGoldAmount();
125             if (inputInfo.aditional > gold)
126                 inputInfo.aditional = gold;
127             depositGold(character.getId(), inputInfo.aditional);
128             character.setGoldAmount(character.getGoldAmount() − inputInfo.aditio
    nal);
129             break;
```



```
130         case InputID::retireGold:
131             gold = retireGold(character.getId(), inputInfo.aditional);
132             character.gainGold(gold);
133             break;
134         default:
135             break;
136     }
137
138 }
139
140 Banker::~Banker() {}
```

jul 21, 20 15:20 **TileSet.h** Page 1/1

```h
#ifndef ARGENTUM_TILESET_H
#define ARGENTUM_TILESET_H

#include <string>
#include <rapidjson/document.h>

class TileSet {
private:
    uint16_t firstgid;
    std::string image;
    uint8_t id;

public:
    explicit TileSet(rapidjson::Value&, uint8_t id);
    TileSet(uint16_t firstGid, uint8_t id);

    virtual ~TileSet();

    int getFirstgid() const;

    const std::string &getImage() const;

    uint8_t getId() const;
};


#endif //ARGENTUM_TILESET_H
```

jul 21, 20 15:20 **TileSet.cpp** Page 1/1

```cpp
#include "TileSet.h"
#include "Identificators.h"

TileSet::TileSet(rapidjson::Value & tilesetDoc, uint8_t id): id(id) {
    firstgid = tilesetDoc["firstgid"].GetInt();
    image = tilesetDoc["image"].GetString();
}

int TileSet::getFirstgid() const {
    return firstgid;
}

const std::string &TileSet::getImage() const {
    return image;
}

uint8_t TileSet::getId() const {
    return id;
}

TileSet::TileSet(uint16_t firstGid, uint8_t id): firstgid(firstGid), id(id) {
    std::string path(ROOT_DIR);
    image = path + "/assets/img/map/" + std::to_string(id) + ".png";
}

TileSet::~TileSet() = default;
```

jul 21, 20 15:20 **TileLayer.h** Page 1/1

```h
#ifndef ARGENTUM_LAYER_H
#define ARGENTUM_LAYER_H

#include <vector>
#include <string>
#include "rapidjson/document.h"

class TileLayer {
private:
    std::vector<uint16_t> data;
    bool isGround;

public:
    TileLayer();
    explicit TileLayer(rapidjson::Value&);
    TileLayer(std::vector<uint16_t> data, bool isGround);

    virtual ~TileLayer();

    const std::vector<uint16_t> &getData() const;

    bool isGroundLayer() const;
};


#endif //ARGENTUM_LAYER_H
```

jul 21, 20 15:20 **TileLayer.cpp** Page 1/1

```cpp
#include "TileLayer.h"

#include <utility>

TileLayer::TileLayer() = default;

TileLayer::~TileLayer() = default;

TileLayer::TileLayer(rapidjson::Value & value) {
    if (value.HasMember("data")) {
        for (auto& a : value["data"].GetArray()) {
            data.push_back(a.GetInt64());
        }
    }
    std::string layerName = value["name"].GetString();
    isGround = "groundLayer" == layerName;
}

const std::vector<uint16_t> &TileLayer::getData() const {
    return data;
}

bool TileLayer::isGroundLayer() const {
    return isGround;
}

TileLayer::TileLayer(std::vector<uint16_t> data, bool isGround): data(std::move(
data)), isGround(isGround) {}
```

jul 21, 20 15:20 **TiledMap.h** Page 1/1

```h
#ifndef ARGENTUM_TILEDMAP_H
#define ARGENTUM_TILEDMAP_H


#include <vector>
#include <string>
#include "TileLayer.h"
#include "TileSet.h"
#include "ObjectLayer.h"

class TiledMap {
private:
    uint16_t width{};
    uint16_t height{};
    uint8_t tileWidth{};
    uint8_t tileHeight{};
    std::vector<TileLayer> tileLayers;
    std::vector<ObjectLayer> objectLayers;
    std::vector<TileSet> tilesets;
public:
    TiledMap();
    explicit TiledMap(rapidjson::Document & json);

    TiledMap(uint16_t width, uint16_t height, uint8_t tileWidth, uint8_t tileHeight,
             std::vector<TileLayer> tileLayers, std::vector<TileSet> tilesets);

    ~TiledMap();

    TiledMap(TiledMap&& other) noexcept ;
    TiledMap& operator=(TiledMap&& other) noexcept ;

    std::vector<ObjectLayer> getObjectLayers();

    const std::vector<TileLayer> &getTileLayers() const;

    const std::vector<TileSet> &getTilesets() const;

    uint16_t getHeight() const;

    uint16_t getWidth() const;

    uint8_t getTileHeight() const;

    uint8_t getTileWidth() const;
};


#endif //ARGENTUM_TILEDMAP_H
```

jul 21, 20 15:20 **TiledMap.cpp** Page 1/2

```cpp
#include "TiledMap.h"

#include <utility>

TiledMap::TiledMap(rapidjson::Document & json) {
    width = json["width"].GetInt();
    height = json["height"].GetInt();
    tileWidth = json["tilewidth"].GetInt();
    tileHeight = json["tileheight"].GetInt();

    rapidjson::Value::Array layers = json["layers"].GetArray();
    for (auto &aLayer : layers) {
        std::string type = aLayer["type"].GetString();
        if ("tilelayer" == type) {
            TileLayer tileSet(aLayer);
            tileLayers.push_back(tileSet);
        } else if ("objectgroup" == type) {
            ObjectLayer objectLayer(aLayer);
            objectLayers.push_back(objectLayer);
        }
    }
    uint8_t idCounter = 0;
    rapidjson::Value::Array tileSetArray = json["tilesets"].GetArray();
    for (auto &aTileSet : tileSetArray) {
        TileSet tileset(aTileSet, ++idCounter);
        tilesets.push_back(tileset);
    }
}

TiledMap::TiledMap() = default;

const std::vector<TileLayer> &TiledMap::getTileLayers() const {
    return tileLayers;
}

std::vector<ObjectLayer> TiledMap::getObjectLayers() {
    return objectLayers;
}

const std::vector<TileSet> &TiledMap::getTilesets() const {
    return tilesets;
}

TiledMap::TiledMap(TiledMap &&other) noexcept {
    std::swap(height, other.height);
    std::swap(width, other.width);
    std::swap(tileHeight, other.tileHeight);
    std::swap(tileWidth, other.tileWidth);
    std::swap(tileLayers, other.tileLayers);
    std::swap(objectLayers, other.objectLayers);
    std::swap(tilesets, other.tilesets);
}

TiledMap &TiledMap::operator=(TiledMap &&other) noexcept {
    if(this == &other) {
        return *this;
    }
    std::swap(height, other.height);
    std::swap(width, other.width);
    std::swap(tileHeight, other.tileHeight);
    std::swap(tileWidth, other.tileWidth);
    std::swap(tileLayers, other.tileLayers);
    std::swap(objectLayers, other.objectLayers);
    std::swap(tilesets, other.tilesets);
    return *this;
}
```

jul 21, 20 15:20 **TiledMap.cpp** Page 2/2

```
67
68  uint16_t TiledMap::getHeight() const {
69      return height;
70  }
71
72  uint16_t TiledMap::getWidth() const {
73      return width;
74  }
75
76  uint8_t TiledMap::getTileHeight() const {
77      return tileHeight;
78  }
79
80  uint8_t TiledMap::getTileWidth() const {
81      return tileWidth;
82  }
83
84  TiledMap::TiledMap(uint16_t width, uint16_t height, uint8_t tileWidth, uint8_t t
    ileHeight,
85                     std::vector<TileLayer> tileLayers, std::vector<TileSet> tiles
    ets) :
86                     width(width), height(height), tileWidth(tileWidth), tileHeigh
    t(tileHeight),
87                     tileLayers(std::move(tileLayers)), tilesets(std::move(tileset
    s)) {}
88
89  TiledMap::~TiledMap() = default;
```

jul 21, 20 15:20 **Thread.h** Page 1/1

```
1   #ifndef THREAD_H
2   #define THREAD_H
3
4   #include <thread>
5
6   class Thread {
7     private:
8       std::thread thread;
9
10    public:
11      Thread();
12      Thread(Thread ∧other);
13      Thread& operator=(Thread∧ other);
14
15      void start();
16      void join();
17      virtual void run() = 0;
18
19      virtual ~Thread();
20
21      Thread(const Thread&) = delete;
22      Thread &operator=(const Thread&) = delete;
23  };
24
25  #endif
```

jul 21, 20 15:20 **Thread.cpp** Page 1/1

```cpp
#include "Thread.h"
#include <utility>

Thread::Thread() = default;

Thread::~Thread() = default;

void Thread::start() {
    this->thread = std::thread(&Thread::run, this);
}

void Thread::join() {
    this->thread.join();
}

Thread::Thread(Thread &&other) {
    this->thread = std::move(other.thread);
}

Thread& Thread::operator=(Thread &&other) {
    this->thread = std::move(other.thread);
    return *this;
}
```

jul 21, 20 15:20 **StaticObject.h** Page 1/1

```cpp
#ifndef ARGENTUM_TALLER_STATICOBJECT_H
#define ARGENTUM_TALLER_STATICOBJECT_H

#include <rapidjson/document.h>
#include "Point.h"

class StaticObject {
private:
    Point topLeft;
    Point topRight;
    Point bottomLeft;
    Point bottomRight;
    std::string name;

public:
    explicit StaticObject(rapidjson::Value &value);

    ~StaticObject();

    const Point &getTopLeft() const;

    const Point &getTopRight() const;

    const Point &getBottomLeft() const;

    const Point &getBottomRight() const;

    const std::string &getName() const;
};


#endif //ARGENTUM_TALLER_STATICOBJECT_H
```

jul 21, 20 15:20 **StaticObject.cpp** Page 1/1

```cpp
#include "StaticObject.h"

StaticObject::~StaticObject() = default;

StaticObject::StaticObject(rapidjson::Value &value) {
    int x = value["x"].GetFloat();
    int y = value["y"].GetFloat();
    int width = value["width"].GetFloat();
    int height = value["height"].GetFloat();
    topLeft = Point(x, y);
    topRight = Point(x + width, y);
    bottomLeft = Point(x, y + height);
    bottomRight = Point(x + width, y + height);
    name = value["name"].GetString();
}

const std::string &StaticObject::getName() const {
    return name;
}

const Point &StaticObject::getTopLeft() const {
    return topLeft;
}

const Point &StaticObject::getTopRight() const {
    return topRight;
}

const Point &StaticObject::getBottomLeft() const {
    return bottomLeft;
}

const Point &StaticObject::getBottomRight() const {
    return bottomRight;
}
```

jul 21, 20 15:20 **Socket.h** Page 1/1

```cpp
#ifndef SOCKET_H
#define SOCKET_H

#include <cstdint>

class Socket {
private:
  int fd;
  //Resuelve la conexiÃ³n con el host (si es indicado) y el puerto al cual el
  //soquet se conecta. El file descriptor que es utilizado queda almacenado
  //en el Socket.
  void resolve_address(struct addrinfo* hints,
            const char* host, const char* port);

  //Contructor solamente utilizado en accept.
  explicit Socket(int fd);

public:
  //Constructores de Socket. Se elimina el constructor por copia.
  //En caso de los constructores por movimiento se invalida a other
  //tomando el ownership del mismo.
  Socket();
  Socket(Socket∧ other);
  Socket& operator=(Socket∧ other);

  //Se encarga de conectar el socket aceptador (del lado del servidor)
  //con el puerto indicado como parÃ¡metro e indicarle la cantidad mÃ¡xima
  //de conexiones en espera que puede haber a la vez. Lanza una
  //SocketException en caso de error.
  void bind_and_listen(const char* port, std::uint32_t max_waiting);

  //Se encarga de conectar el cliente con el (host, port) indicando como
  //parÃ¡metro. Lanza una SocketException en caso de error.
  void connect(const char* host, const char* port);

  //Acepta la conexiÃ³n de un cliente. Devuelve al Socket por movimiento.
  //Lanza una SocketException en caso de error.
  Socket accept();

  //EnvÃ-a un stream que comienza en buffer de longitud length.
  //Devuelve la cantidad de bytes enviados.
  //Lanza una SocketException en caso de error.
  int send(const void* buffer, std::uint32_t length) const;

  //Almacena en un buffer de longitud length todos los bytes
  //recibidos en el socket self. devuelve la cantidad de bytes recibidos.
  //Lanza una SocketException en caso de error.
  int recieve(void* buffer, std::uint32_t length) const;

  //channel: SHUT_WR, SHUT_RD, SHUT_RDWR
  void shutdown(int channel);

  //Realiza un cierre del Socket, invalidandolo.
  void close();

  ~Socket();

  Socket(const Socket& other) = delete;
  Socket& operator=(const Socket& other) = delete;
};

#endif
```

jul 21, 20 15:20 **SocketException.h** Page 1/1

```h
#ifndef SOCKETEXCEPTION_H
#define SOCKETEXCEPTION_H

#include <typeinfo>
#define BUFF_SIZE 256

class SocketException : public std::exception {
private:
  char msg_error[BUFF_SIZE];
public:
  explicit SocketException(const char* msg, ...) noexcept;

  virtual const char* what() const noexcept;

  virtual~SocketException() noexcept;
};

#endif
```

jul 21, 20 15:20 **SocketException.cpp** Page 1/1

```cpp
#include "SocketException.h"
#include <errno.h>
#include <cstdio>
#include <cstring>
#include <cstdarg>

SocketException::SocketException(const char* msg, ...) noexcept {
  int _errno = errno;

  va_list args;
  va_start(args, msg);
  int s = vsnprintf(msg_error, BUFF_SIZE, msg, args);
  va_end(args);

  strncpy(msg_error+s, strerror(_errno), BUFF_SIZE-s);
  msg_error[BUFF_SIZE-1] = 0;
}

const char* SocketException::what() const noexcept {
  return msg_error;
}

SocketException::~SocketException() noexcept {}
```

jul 21, 20 15:20 **Socket.cpp** Page 1/3

```cpp
#include "Socket.h"
#include "SocketException.h"
#include <utility>
#include <cstring>
#include <unistd.h>
#include <stdexcept>
#include <netdb.h>
#include <sys/socket.h>

#define FAIL_FD "No se pudo conectar a ningÃºn file descriptor."
#define FAIL_GETADRR "Error en getaddrinfo."
#define FAIL_BIND "Error al realizar el bind al puerto %s."
#define FAIL_CONNECT "Error al conectarse al host %s y puerto %s."
#define FAIL_ACCEPT "Error al intentar aceptar un nuevo cliente."
#define FAIL_SEND "Error al enviar desde el socket %d."
#define FAIL_RECV "Error al recibir desde el socket %d."

Socket::Socket() : fd(-1) {}

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

void Socket::resolve_address(struct addrinfo* hints,
                             const char* host, const char* port) {
  struct addrinfo *results, *iter;
  int val = 1;
  if (getaddrinfo(host, port, hints, &results) ≠ 0) {
    freeaddrinfo(results);
    throw SocketException(FAIL_GETADRR);
  }

  for (iter = results; iter ≠ nullptr; iter = iter→ai_next) {
    this→fd = ::socket(iter→ai_family,iter→ai_socktype,0);

    if (this→fd ≡ -1)
      continue;

    if(host ≡ nullptr) {
      if (::bind(this→fd, iter→ai_addr, iter→ai_addrlen) ≡ 0) {
        setsockopt(this→fd, SOL_SOCKET,SO_REUSEADDR, &val, sizeof(val));
        break;
      }
    } else {
      if(::connect(this→fd, iter→ai_addr, iter→ai_addrlen) ≠ -1)
        break;
    }
    ::close(this→fd);
    this→fd = -1;
  }

  freeaddrinfo(results);
  if(this→fd ≡ -1)
    throw SocketException(FAIL_FD);
  return;
}

void Socket::bind_and_listen(const char* port, std::uint32_t max_waiting) {
  struct addrinfo hints;

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET; //IPv4
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;
  try{
    resolve_address(&hints, nullptr, port);
  } catch (const SocketException& e) {
    throw SocketException(FAIL_BIND, port);
  }
  ::listen(this→fd, max_waiting);

  return;
}

void Socket::connect(const char* host, const char* port) {
  struct addrinfo hints;

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET; //IPv4
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = 0;
  try{
    resolve_address(&hints, host, port);
  } catch (const SocketException& e) {
    throw SocketException(FAIL_CONNECT, host, port);
  }
  return;
}

Socket Socket::accept() {
  int new_fd = ::accept(this→fd, nullptr, nullptr);
  if (new_fd ≡ -1)
    throw SocketException(FAIL_ACCEPT);
  return std::move(Socket(new_fd));
}

int Socket::send(const void* buffer, std::uint32_t length) const {
  std::uint32_t sended_bytes = 0;
  int result_send;
  std::uint32_t remaining_bytes = length;
  const char* char_buffer = (const char*) buffer;
  while(sended_bytes < length) {
    result_send = ::send(this→fd, &char_buffer[sended_bytes],
                         remaining_bytes, MSG_NOSIGNAL);
    if(result_send ≡ -1 ∨ result_send ≡ 0)
      throw SocketException(FAIL_SEND, this→fd);
    sended_bytes += result_send;
    remaining_bytes -= result_send;
  }
  return sended_bytes;
}

int Socket::recieve(void* buffer, std::uint32_t length) const {
  std::uint32_t received_bytes = 0;
  int result_recv;
  std::uint32_t remaining_bytes = length;
  char* char_buffer = (char*) buffer;

  while(received_bytes < length) {
    result_recv = ::recv(this→fd, &char_buffer[received_bytes],
                         remaining_bytes, 0);
    if(result_recv ≡ -1 ∨ result_recv ≡ 0)
      throw SocketException(FAIL_RECV, this→fd);
    received_bytes += result_recv;
    remaining_bytes -= result_recv;
  }
  return received_bytes;
}

Socket::Socket(Socket∧ other) : fd(other.fd) {
  other.fd = -1;
}

Socket& Socket::operator=(Socket∧ other) {
```

jul 21, 20 15:20 **Socket.cpp** Page 2/3

jul 21, 20 15:20 **Socket.cpp** Page 3/3

```
133   this→fd = other.fd;
134   other.fd = -1;
135   return *this;
136 }
137
138 void Socket::shutdown(int channel) {
139   ::shutdown(this→fd, channel);
140 }
141
142 void Socket::close() {
143   if (this→fd ≠ -1){
144     shutdown(SHUT_RDWR);
145     ::close(this→fd);
146   }
147   this→fd = -1;
148 }
149
150 Socket::~Socket() {
151   this→close();
152 }
```

jul 21, 20 15:20 **Random.h** Page 1/1

```
1  #ifndef ARGENTUM_TALLER_RANDOM_H
2  #define ARGENTUM_TALLER_RANDOM_H
3
4
5  class Random {
6  public:
7      static int get(int minRand, int maxRand);
8      static float getFloat(float minRand, float maxRand);
9  };
10
11
12 #endif //ARGENTUM_TALLER_RANDOM_H
```

jul 21, 20 15:20 **Random.cpp** Page 1/1

```cpp
#include "Random.h"
#include <random>

float Random::getFloat(float minRand, float maxRand) {
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<float> dist(minRand, maxRand);
    return dist(mt);
}

int Random::get(int minRand, int maxRand) {
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_int_distribution<int> dist(minRand, maxRand);
    return dist(mt);
}
```

jul 21, 20 15:20 **Point.h** Page 1/1

```cpp
#ifndef POINT_H
#define POINT_H
#include <cmath>
class Point {
public:
    float x,y;

    explicit Point(float x=0, float y=0);

    Point& operator=(const Point& p2);

    friend Point operator+(Point p1, const Point& p2);

    Point& operator +=(const Point& p2);

    friend Point operator−(Point p1, const Point& p2);

    Point& operator−=(const Point& p2);

    friend Point operator*(const Point& p1, float n);

    friend bool operator≡(Point p1, const Point& p2);

    friend bool operator≠(Point p1, const Point& p2);

    float distance(const Point& p2);
};

#endif
```

jul 21, 20 15:20 **Point.cpp** Page 1/1

```cpp
#include "Point.h"

#include <cmath>

Point::Point(float x, float y) : x(x), y(y) {}

Point& Point::operator=(const Point& p2) {
    this→x = p2.x;
    this→y = p2.y;
    return *this;
}

Point operator+(Point p1,const Point& p2) {
    return Point(p1.x+p2.x, p1.y+p2.y);
}

Point& Point::operator+=(const Point& p2) {
    this→x += p2.x;
    this→y += p2.y;
    return *this;
}

Point& Point::operator−=(const Point& p2) {
    this→x −= p2.x;
    this→y −= p2.y;
    return *this;
}

Point operator−(Point p1,const Point& p2) {
    return Point(p1.x−p2.x, p1.y−+p2.y);
}

bool operator≡(Point p1,const Point& p2) {
    return (p1.x≡p2.x) ∧ (p1.y ≡ p2.y);
}

Point operator*(const Point& p1,float n) {
    return Point(p1.x*n,p1.y*n);
}

bool operator≠(Point p1, const Point& p2) {
    return ¬(p1≡p2);
}

float Point::distance(const Point& p2) {
    Point dist = *this − p2;
    return std::sqrt(dist.x * dist.x + dist.y * dist.y);
}
```

jul 21, 20 15:20 **PlayerInfo.h** Page 1/1

```cpp
#ifndef ARGENTUM_TALLER_PLAYERINFO_H
#define ARGENTUM_TALLER_PLAYERINFO_H

#include <zconf.h>
#include <string>
#include "GameObjectInfo.h"
#include "Identificators.h"

class PlayerInfo : public GameObjectInfo{
private:
    uint goldAmount{};
    uint life{};
    uint mana{};
    uint level{};
    uint exp{};
    uint maxLife{};
    uint maxMana{};
    uint maxExp{};
    uint safeGold{};
    std::string inventory;
    std::string name;

public:
    PlayerInfo();
    PlayerInfo(uint id, Point point, uint goldAmount, uint life, uint mana,
        const std::string& textureHashId, Direction direction,uint safeGold,
        uint maxLife, uint maxMana, uint exp, uint maxExp, uint level,
        std::string  inventory, CharacterStateID state, WeaponID attackBy);

    ~PlayerInfo() override;

    uint getGoldAmount() const;

    uint getLife() const;

    uint getMana() const;

    Direction getDirection() const;

    uint getExp() const;

    uint getLevel() const;

    uint getSafeGold() const;

    uint getMaxLife() const;

    uint getMaxMana() const;

    uint getMaxExp() const;

    std::string getName() const;

    std::string getInventory() const;
};


#endif //ARGENTUM_TALLER_PLAYERINFO_H
```

jul 21, 20 15:20 **PlayerInfo.cpp** Page 1/2

```cpp
#include "PlayerInfo.h"

#include <utility>

PlayerInfo::~PlayerInfo() = default;

uint PlayerInfo::getGoldAmount() const {
    return goldAmount;
}

uint PlayerInfo::getLife() const {
    return life;
}

uint PlayerInfo::getMana() const {
    return mana;
}

PlayerInfo::PlayerInfo(uint id, Point point, uint goldAmount, uint life, uint mana, const std::string& textureHashId,
    Direction direction, uint safeGold, uint maxLife, uint maxMana, uint exp, uint maxExp,
    uint level,std::string  inventory, CharacterStateID state, WeaponID attackBy)
    :GameObjectInfo(id, point, textureHashId, direction,state, false ,attackBy),
    goldAmount(goldAmount), life(life), mana(mana), level(level), exp(exp), maxLife(maxLife),
    maxMana(maxMana), maxExp(maxExp), safeGold(safeGold) , inventory(std::move(inventory)) {}

PlayerInfo::PlayerInfo() : GameObjectInfo() {}

Direction PlayerInfo::getDirection() const {
    return this→direction;
}

uint PlayerInfo::getMaxExp() const {
    return this→maxExp;
}

uint PlayerInfo::getMaxLife() const {
    return this→maxLife;
}

uint PlayerInfo::getMaxMana() const {
    return this→maxMana;
}

uint PlayerInfo::getSafeGold() const {
    return this→safeGold;
}

uint PlayerInfo::getExp() const {
    return this→exp;
}

uint PlayerInfo::getLevel() const {
    return this→level;
}

std::string PlayerInfo::getName() const {
    return this→name;
}

std::string PlayerInfo::getInventory() const {
    return this→inventory;
```

jul 21, 20 15:20 **PlayerInfo.cpp** Page 2/2

```cpp
}
```

jul 21, 20 15:20 **ObjectLayer.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_OBJECTLAYER_H
#define ARGENTUM_TALLER_OBJECTLAYER_H

#include <string>
#include <vector>
#include <rapidjson/pointer.h>
#include "StaticObject.h"

class ObjectLayer {
private:
    std::string name;
    std::vector<StaticObject> objects;

public:
    explicit ObjectLayer(rapidjson::Value&);

    ~ObjectLayer();

    const std::string &getName() const;

    std::vector<StaticObject> getObjects();
};


#endif //ARGENTUM_TALLER_OBJECTLAYER_H
```

jul 21, 20 15:20 **ObjectLayer.cpp** Page 1/1

```cpp
#include "ObjectLayer.h"

ObjectLayer::ObjectLayer(rapidjson::Value &value) {
    name = value["name"].GetString();
    rapidjson::Value::Array objectsArray = value["objects"].GetArray();
    for (auto &objectValue : objectsArray) {
        StaticObject mapObject(objectValue);
        objects.push_back(std::move(mapObject));
    }
}

ObjectLayer::~ObjectLayer() = default;

const std::string &ObjectLayer::getName() const {
    return name;
}

std::vector<StaticObject> ObjectLayer::getObjects() {
    return objects;
}
```

jul 21, 20 15:20 **Message.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_MESSAGE_H
#define ARGENTUM_TALLER_MESSAGE_H


#include <vector>
#include <cstdint>

//Wrapper del mensaje recibido por el socket.
//Se puede obtener la longitud del mismo, su tipo de mensaje
//y la informaciÃ³n propiamente del mensaje enviado.
//AdemÃ¡s provee un mecanismo de lectura de distintos bytes.
class Message {
private:
    std::vector<uint8_t> data;
    uint32_t length{};
    uint8_t type{};
    uint32_t pos = 0;
public:
    Message();

    Message(std::vector<uint8_t>& data, uint32_t length, uint8_t type);

    ~Message();

    uint8_t getType() const;

    void clear();

    uint8_t read8();

    uint16_t read16();

    uint32_t read32();
};

#endif //ARGENTUM_TALLER_MESSAGE_H
```

jul 21, 20 15:20 **Message.cpp** Page 1/2

```cpp
#include "Message.h"

#include <utility>
#include "Exception.h"

#define ERRORREAD "Se quiere ingresar a una posicion no valida"

Message::~Message() = default;

Message::Message() = default;

uint8_t Message::getType() const {
    return type;
}

Message::Message(std::vector<uint8_t>& data, uint32_t length, uint8_t type) :
data(std::move(data)), length(length), type(type) {}

void Message::clear() {
    pos = 0;
}

uint32_t conversorTo32(const uint8_t* value) {
    uint8_t temp[4];
    uint32_t* temp32;
    for (int j = 0; j < 4; j++) {
        temp[j] = value[j];
    }
    temp32 = (uint32_t*) temp;
    return *temp32;
}

uint16_t conversorTo16(const uint8_t* value) {
    uint8_t temp[4];
    uint16_t* temp16;
    for (int j = 0; j < 2; j++) {
        temp[j] = value[j];
    }
    temp16 = (uint16_t*) temp;
    return *temp16;
}

uint8_t Message::read8() {
    if (pos > length) {
        throw Exception(ERRORREAD);
    }
    uint8_t value = *(data.data() + pos);
    pos++;
    return value;
}

uint16_t Message::read16() {
    if (pos + 2 > length) {
        throw Exception(ERRORREAD);
    }
    uint16_t value = conversorTo16(data.data() + pos);
    pos += 2;
    return value;
}

uint32_t Message::read32() {
    if (pos + 4 > length) {
        throw Exception(ERRORREAD);
    }
    uint32_t value = conversorTo32(data.data() + pos);
    pos += 4;
```

jul 21, 20 15:20 **Message.cpp** Page 2/2

```
67      return value;
68  }
```

jul 21, 20 15:20 **JsonReader.h** Page 1/1

```
1   #ifndef ARGENTUM_JSONREADER_H
2   #define ARGENTUM_JSONREADER_H
3
4
5   #include <rapidjson/document.h>
6
7   class JsonReader {
8   public:
9       JsonReader();
10
11      virtual ~JsonReader();
12
13      static rapidjson::Document read(const std::string&);
14
15  };
16
17
18  #endif //ARGENTUM_JSONREADER_H
```

jul 21, 20 15:20 **JsonReader.cpp** Page 1/1

```cpp
#include <fstream>
#include <rapidjson/istreamwrapper.h>
#include "JsonReader.h"
#include <rapidjson/error/en.h>
#include "Exception.h"

JsonReader::JsonReader() = default;

JsonReader::~JsonReader() = default;

rapidjson::Document JsonReader::read(const std::string & filename) {
    std::ifstream file(filename);
    rapidjson::IStreamWrapper isw(file);
    rapidjson::Document json;
    json.ParseStream(isw);
    if (json.HasParseError()) {
        throw Exception("There are some problems with parsing", rapidjson::GetParseError_En(json.GetParseError()));
    }
    return json;
}
```

jul 21, 20 15:20 **InputQueue.h** Page 1/1

```cpp
#ifndef INPUTQUEUE_H
#define INPUTQUEUE_H

#include "BlockingQueue.h"
#include "Identificators.h"

using InputQueue = BlockingQueue<InputInfo>;

#endif
```

jul 21, 20 15:20 **Identificators.h** Page 1/4

```
#ifndef CHARACTERSTATESID_H
#define CHARACTERSTATESID_H

#include "Point.h"
#include <vector>
#include <unordered_map>
#include <iostream>

#ifdef DEV
#define ROOT_DIR "./"
#define CONFIG_DIR "./json"
#else
#define ROOT_DIR "/var/Argentum"
#define CONFIG_DIR "/etc/Argentum"
#endif

enum class CharacterStateID {
    Still, //Aplica a NPCServer
    Move,  //Aplica a NPCServer
    Attack, //Aplica a NPCServer
    Meditate,
    Interact,
    Resurrect,
};

enum class InputID {
    nothing,
    stopMove,
    up, //w
    down, //s
    left, //a
    right, //d
    meditate, //q
    resurrect, //r
    cure, //h
    buy,
    sell,
    depositItem,
    retireItem,
    depositGold,
    retireGold,
    selectTarget, //click
    equipItem,
    dropItem,
    takeItem, //e
    unequipItem,
};

enum class HelmetID {
    Nothing,
    Hood,
    IronHelmet,
    MagicHat,
};

enum class BodyID {
    Nothing,
    RedCommon,
    BlueCommon,
    GreenCommon,
    BlueTunic,
    LeatherArmor,
    PlateArmor,
    Ghost,
    Banker,
    Merchant,
```

jul 21, 20 15:20 **Identificators.h** Page 2/4

```
    Priest,
    Goblin,
    Skeleton,
    Spider,
    Zombie,
};

enum class HeadID {
    Nothing,
    Human,
    Elf,
    Dwarf,
    Gnome,
    Priest,
    Zombie,
};

enum class ShieldID {
    Nothing,
    TurtleShield,
    IronShield,
};

enum class WeaponID {
    Nothing,
    SimpleArc,
    CompoundArc,
    LongSword,
    Hammer,
    Ax,
    ElficFlaute,
    AshStick,
    GnarledStick,
    Crosier,
};

enum class ItemsInventoryID {
    Nothing,
    SimpleArc,
    CompoundArc,
    LongSword,
    Hammer,
    Ax,
    ElficFlaute,
    AshStick,
    GnarledStick,
    Crosier,
    TurtleShield,
    IronShield,
    Hood,
    IronHelmet,
    MagicHat,
    RedCommon,
    BlueCommon,
    GreenCommon,
    BlueTunic,
    LeatherArmor,
    PlateArmor,
    HealthPotion,
    ManaPotion,
    Gold,
};

enum class Direction{
    down,
    up,
```

```
    left,
    right,
};

enum class RaceID {
    Nothing,
    Human,
    Elf,
    Dwarf,
    Gnome
};

enum class GameClassID {
    Nothing,
    Mage,
    Cleric,
    Paladin,
    Warrior
};

enum class CreatureID {
    Nothing,
    Zombie,
    Skeleton,
    Spider,
    Goblin,
};

enum class ActionsProfessionID {
    Nothing,
    DepositItem,
    DepositGold,
    RetireItem,
    RetireGold,
    Buy,
    Sell,
    Resurrect,
    Cure,
};

struct NPCInfo {
    uint8_t type = 0;
    std::vector<ActionsProfessionID> actions;
    std::unordered_map<ItemsInventoryID,uint> items;
    uint32_t gold;
    std::vector<ItemsInventoryID> itemsInBank;
};

struct InputInfo {
    InputID input = InputID::nothing;
    Point position;
    int aditional = 0;
};

enum class StateID {
    Still,
    Move,
    Equip,
    Transition,
    Interact,
    Attack,
    Pursuit,
    Meditate,
    TakeDrop,
    Resurrect,
};
```



```

#endif
```

jul 21, 20 15:20 **GameObjectInfo.h** Page 1/1

```h
#ifndef ARGENTUM_TALLER_GAMEOBJECTINFO_H
#define ARGENTUM_TALLER_GAMEOBJECTINFO_H


#include <zconf.h>
#include <string>
#include "Identificators.h"

class GameObjectInfo {
protected:
    uint id{};
    Point point;
    Direction direction;
    std::string textureHashId;
    CharacterStateID state;
    bool item;
    WeaponID attackBy;
public:
    GameObjectInfo();
    GameObjectInfo(uint id, const Point &point, std::string textureHashId, Direction direction,
        CharacterStateID state, bool item, WeaponID attackBy);

    const Point &getPoint() const;

    const std::string &getTextureHashId() const;

    virtual ~GameObjectInfo();

    uint getId() const;

    float getX() const;

    float getY() const;

    Direction getDirection() const;

    HelmetID getHelmetID() const;
    HeadID getHeadID() const;
    BodyID getBodyID() const;
    WeaponID getWeaponID() const;
    ShieldID getShieldID() const;
    ItemsInventoryID getItemID() const;
    CharacterStateID getState() const;
    WeaponID getAttackWeapon() const;

    bool isItem() const;
};


#endif //ARGENTUM_TALLER_GAMEOBJECTINFO_H
```

jul 21, 20 15:20 **GameObjectInfo.cpp** Page 1/2

```cpp
#include "GameObjectInfo.h"
#include <utility>

GameObjectInfo::~GameObjectInfo() = default;

GameObjectInfo::GameObjectInfo(uint id, const Point &point, std::string textureHashId, Direction aDirection,
     CharacterStateID state, bool item, WeaponID attackBy) : id(id), point(point), direction(aDirection),
      textureHashId(std::move(textureHashId)), state(state), item(item), attackBy(attackBy) {}

uint GameObjectInfo::getId() const {
    return id;
}

const Point &GameObjectInfo::getPoint() const {
    return point;
}

const std::string &GameObjectInfo::getTextureHashId() const {
    return textureHashId;
}

float GameObjectInfo::getX() const {
    return point.x;
}

float GameObjectInfo::getY() const {
    return point.y;
}

HelmetID GameObjectInfo::getHelmetID() const {
    std::string stringId = this→textureHashId.substr(2,2);
    int id = std::stoi(stringId);
    return (HelmetID)id;
}

HeadID GameObjectInfo::getHeadID() const {
    std::string stringId = this→textureHashId.substr(6,2);
    int id = std::stoi(stringId);
    return (HeadID)id;
}

BodyID GameObjectInfo::getBodyID() const {
    std::string stringId = this→textureHashId.substr(10,2);
    int id = std::stoi(stringId);
    return (BodyID)id;
}
WeaponID GameObjectInfo::getWeaponID() const {
    std::string stringId = this→textureHashId.substr(18,2);
    int id = std::stoi(stringId);
    return (WeaponID)id;
}
ShieldID GameObjectInfo::getShieldID() const {
    std::string stringId = this→textureHashId.substr(14,2);
    int id = std::stoi(stringId);
    return (ShieldID)id;
}

ItemsInventoryID GameObjectInfo::getItemID() const {
    std::string stringId = this→textureHashId.substr(22,2);
    int id = std::stoi(stringId);
    return (ItemsInventoryID)id;
}

```

jul 21, 20 15:20 **GameObjectInfo.cpp** Page 2/2

```
64  Direction GameObjectInfo::getDirection() const {
65      return direction;
66  }
67
68  bool GameObjectInfo::isItem() const {
69      return this→item;
70  }
71
72  CharacterStateID GameObjectInfo::getState() const {
73      return this→state;
74  }
75
76  WeaponID GameObjectInfo::getAttackWeapon() const {
77      return this→attackBy;
78  }
79
80  GameObjectInfo::GameObjectInfo() = default;
```

jul 21, 20 15:20 **Exception.h** Page 1/1

```
1   #ifndef EXCEPTION_H
2   #define EXCEPTION_H
3
4   #include <typeinfo>
5   #define BUFF_SIZE 256
6
7   class Exception : public std::exception {
8   private:
9     char msg_error[BUFF_SIZE];
10  public:
11    explicit Exception(const char* msg, ...) noexcept;
12
13    virtual const char* what() const noexcept;
14
15    virtual ~Exception() noexcept;
16  };
17
18  #endif
```

jul 21, 20 15:20 **Exception.cpp** Page 1/1

```cpp
#include "Exception.h"
#include <errno.h>
#include <cstdio>
#include <cstring>
#include <cstdarg>

Exception::Exception(const char* msg, ...) noexcept {
  int _errno = errno;

  va_list args;
  va_start(args, msg);
  int s = vsnprintf(msg_error, BUFF_SIZE, msg, args);
  va_end(args);

  strncpy(msg_error+s, strerror(_errno), BUFF_SIZE-s);
  msg_error[BUFF_SIZE-1] = 0;
}

const char* Exception::what() const noexcept {
  return msg_error;
}

Exception::~Exception() noexcept {}
```

jul 21, 20 15:20 **Decoder.h** Page 1/1

```cpp
#ifndef ARGENTUM_TALLER_DECODER_H
#define ARGENTUM_TALLER_DECODER_H


#include <cstdint>
#include <vector>
#include "PlayerInfo.h"
#include "TiledMap.h"
#include "Message.h"

//Este protocolo binario de comunicaciÃ³n estÃ¡ diseÃ±ado para enviar el mapa, la
//informaciÃ³n del jugador, los objetos renderizables del mapa(items, criaturas, npc u otros jugadores),
//la informaciÃ³n del NPC con el que interactua y los comandos que ingresa el jugador.

class Decoder {
private:
    static void encodeStatsPlayer(const PlayerInfo &info, std::vector<uint8_t>&);
    static void encodeEquipmentPlayer(const PlayerInfo &info, std::vector<uint8_t>&);
    static void encodeInventory(const PlayerInfo &info, std::vector<uint8_t>&);
    static void encodeStatePlayer(const GameObjectInfo &info, std::vector<uint8_t>&);
    static void conversorTo8(uint32_t value, uint8_t from, std::vector<uint8_t>& encodeMsg);
    static std::string decodeEquipment(Message& msg, bool isGameObject=false);
    static std::string decodeInventory(Message& msg);
    static void encodeItem(GameObjectInfo object, std::vector<uint8_t>&);
    static void encodeCharacter(GameObjectInfo object, std::vector<uint8_t>&);
public:
    Decoder();
    //De un PlayerInfo genera una codificaciÃ³n correspondiente
    static std::vector<uint8_t> encodePlayerInfo(const PlayerInfo &info);

    static std::vector<uint8_t> encodeCommand(InputInfo input);

    static std::vector<uint8_t> encodeInit(RaceID race, GameClassID gameClass);

    static std::vector<uint8_t> encodeMap(const TiledMap&);

    static std::vector<uint8_t> encodeNPCInfo(const NPCInfo& info);

    static std::vector<uint8_t> encodeGameObjects(const std::vector<GameObjectInfo> &objects);

    //Se recibe como parÃ¡metro el mensaje recibido desde el servidor
    //Devuelve un PlayerInfo con toda la informaciÃ³n del jugador
    static PlayerInfo decodePlayerInfo(Message msg);

    static InputInfo decodeCommand(Message msg);

    static std::vector<GameObjectInfo> decodeGameObjects(Message msg);

    static TiledMap decodeMap(Message msg);

    static NPCInfo decodeNPCInfo(Message msg);

    virtual ~Decoder();
};


#endif //ARGENTUM_TALLER_DECODER_H
```

```cpp
#include <netinet/in.h>
#include "Decoder.h"

#define ZERO 0x00
#define MAPMSG 0x00
#define PLAYERINFOMSG 0x01
#define OBJECTSINFOMSG 0x02
#define COMMANDMSG 0x03
#define INITMSG 0x04
#define INTERACTMSG 0x05
#define INITLENGTH 2
#define OBJECTLENGTH 18
#define PLAYERINFOLENGTH 47
#define COMMANDLENGTH 7
#define MAPLENGTH 240046
#define ITEM 0x01
#define CHARACTER 0x00
#define LIMITINVENTORY 9


Decoder::Decoder() = default;

void Decoder::conversorTo8(uint32_t value, uint8_t from, std::vector<uint8_t>& encodeMsg) {
    int max = 0;
    if (from == 16) {
        max = 2;
    } else if (from == 32) {
        max = 4;
    }
    auto* ptr = (uint8_t*) &value;
    for(int i = 0; i < max; i++)
        encodeMsg.push_back(*(ptr+i));
}

void Decoder::encodeStatsPlayer(const PlayerInfo &info, std::vector<uint8_t>& encodeMsg) {
    uint16_t life,maxLife, mana, maxMana, level, gold, safeGold;
    uint32_t maxExp, exp;
    life = htons(info.getLife());
    maxLife = htons(info.getMaxLife());
    mana = htons(info.getMana());
    maxMana = htons(info.getMaxMana());
    exp = htonl(info.getExp());
    maxExp = htonl(info.getMaxExp());
    level = htons(info.getLevel());
    gold = htons(info.getGoldAmount());
    safeGold = htons(info.getSafeGold());
    conversorTo8(life, 16, encodeMsg);
    conversorTo8(maxLife, 16, encodeMsg);
    conversorTo8(mana, 16, encodeMsg);
    conversorTo8(maxMana, 16, encodeMsg);
    conversorTo8(exp, 32, encodeMsg);
    conversorTo8(maxExp, 32, encodeMsg);
    conversorTo8(level, 16, encodeMsg);
    conversorTo8(gold, 16, encodeMsg);
    conversorTo8(safeGold, 16, encodeMsg);
}

void Decoder::encodeInventory(const PlayerInfo &info, std::vector<uint8_t>& encodeMsg) {
    std::string inventory = info.getInventory();
    std::string item;
    uint8_t idItem;
    for (int i = 0; i<LIMITINVENTORY; i++) {
        item = inventory.substr(2*i+i,2);
```



```cpp
        idItem = std::stoi(item);
        encodeMsg.push_back(idItem);
    }
}

void Decoder::encodeEquipmentPlayer(const PlayerInfo &info, std::vector<uint8_t>& encodeMsg) {
    uint8_t idHelmet = (uint8_t) info.getHelmetID();
    uint8_t idHead = (uint8_t) info.getHeadID();
    uint8_t idBody = (uint8_t) info.getBodyID();
    uint8_t idShield = (uint8_t) info.getShieldID();
    uint8_t idWeapon = (uint8_t) info.getWeaponID();

    encodeMsg.push_back(idHelmet);
    encodeMsg.push_back(idHead);
    encodeMsg.push_back(idBody);
    encodeMsg.push_back(idShield);
    encodeMsg.push_back(idWeapon);
}

void Decoder::encodeStatePlayer(const GameObjectInfo &info, std::vector<uint8_t>& encodeMsg) {
    uint16_t state = htons((uint16_t) info.getState());
    uint16_t dir = htons((uint16_t) info.getDirection());
    uint16_t posX = htons(info.getPoint().x);
    uint16_t posY = htons(info.getPoint().y);
    conversorTo8(state, 16, encodeMsg);
    conversorTo8(dir, 16, encodeMsg);
    conversorTo8(posX, 16, encodeMsg);
    conversorTo8(posY, 16, encodeMsg);
}

std::vector<uint8_t> Decoder::encodePlayerInfo(const PlayerInfo &info) {
    std::vector<uint8_t> encodeMsg;
    uint32_t length = PLAYERINFOLENGTH;
    length = htonl(length);
    conversorTo8(length, 32, encodeMsg);
    uint8_t type = PLAYERINFOMSG;
    encodeMsg.push_back(type);
    uint16_t id = info.getId();
    id = htons(id);
    conversorTo8(id, 16, encodeMsg);
    encodeStatsPlayer(info, encodeMsg);
    encodeEquipmentPlayer(info, encodeMsg);
    encodeInventory(info, encodeMsg);
    encodeStatePlayer(info, encodeMsg);
    uint8_t attackBy = (uint8_t) info.getAttackWeapon();
    encodeMsg.push_back(attackBy);
    return encodeMsg;
}

std::string Decoder::decodeEquipment(Message& msg, bool isGameObject) {
    std::string equipment;
    uint8_t id;
    std::string temp;
    equipment += "ht";
    id = msg.read8();
    temp = std::to_string(id);
    if (temp.size() == 1)
        equipment += "0";
    equipment += temp + "|";
    equipment += "h";
    id = msg.read8();
    temp = std::to_string(id);
    if (temp.size() == 1)
        equipment += "0";
```

```
    equipment += temp + "|";
    equipment += "b";
    id = msg.read8();
    temp = std::to_string(id);
    if (temp.size() == 1)
        equipment += "0";
    equipment += temp + "|";
    equipment += "s";
    id = msg.read8();
    temp = std::to_string(id);
    if (temp.size() == 1)
        equipment += "0";
    equipment += temp + "|";
    equipment += "w";
    id = msg.read8();
    temp = std::to_string(id);
    if (temp.size() == 1)
        equipment += "0";
    equipment += temp;
    if(isGameObject) {
        equipment +="|i";
        id = msg.read8();
        temp = std::to_string(id);
        if (temp.size() == 1)
            equipment += "0";
        equipment += temp;
    }
    return equipment;
}

std::string Decoder::decodeInventory(Message& msg) {
    std::string inventory;
    uint8_t id;
    std::string temp;
    for (int i=0; i<LIMITINVENTORY;i++) {
        id = msg.read8();
        temp = std::to_string(id);
        if (temp.size() == 1)
            inventory += "0";
        inventory += temp;
        if (i != LIMITINVENTORY-1)
            inventory += "|";
    }
    return inventory;
}

PlayerInfo Decoder::decodePlayerInfo(Message msg) {
    msg.clear();
    uint16_t id = ntohs(msg.read16());
    uint16_t life = ntohs(msg.read16());
    uint16_t maxLife = ntohs(msg.read16());
    uint16_t mana = ntohs(msg.read16());
    uint16_t maxMana = ntohs(msg.read16());
    uint32_t exp = ntohl(msg.read32());
    uint32_t maxExp = ntohl(msg.read32());
    uint16_t level = ntohs(msg.read16());
    uint16_t gold = ntohs(msg.read16());
    uint16_t safeGold = ntohs(msg.read16());
    std::string equipment = decodeEquipment(msg, false);
    std::string inventory = decodeInventory(msg);
    auto state = (CharacterStateID) (ntohs(msg.read16()));
    auto dir = (Direction) (ntohs(msg.read16()));
    uint16_t x = ntohs(msg.read16());
    uint16_t y = ntohs(msg.read16());
    auto attackBy = (WeaponID) msg.read8();
    PlayerInfo info(id, Point(x,y), gold, life, mana, equipment, dir,
```



```
                    safeGold,maxLife,maxMana,exp,maxExp,level,inventory,state,attackBy);
    return info;
}

void Decoder::encodeItem(GameObjectInfo object, std::vector<uint8_t>& encodeMsg) {
    encodeMsg.push_back(ITEM);
    auto idHelmet = (uint8_t) object.getHelmetID();
    auto idHead = (uint8_t) object.getHeadID();
    auto idBody = (uint8_t) object.getBodyID();
    auto idShield = (uint8_t) object.getShieldID();
    auto idWeapon = (uint8_t) object.getWeaponID();
    auto idItem = (uint8_t) object.getItemID();

    encodeMsg.push_back(idHelmet);
    encodeMsg.push_back(idHead);
    encodeMsg.push_back(idBody);
    encodeMsg.push_back(idShield);
    encodeMsg.push_back(idWeapon);
    encodeMsg.push_back(idItem);
    //Encodeo el estado y direccion del item con 0 dado que no me interesa su valor
    for (int i=0; i<4;i++){
        encodeMsg.push_back(ZERO);
    }

    uint16_t posX = htons(object.getPoint().x);
    uint16_t posY = htons(object.getPoint().y);
    conversorTo8(posX,16, encodeMsg);
    conversorTo8(posY,16, encodeMsg);
    encodeMsg.push_back(ZERO); //Un item no puede ser atacado
}

void Decoder::encodeCharacter(GameObjectInfo object, std::vector<uint8_t>& encodeMsg) {
    encodeMsg.push_back(CHARACTER);
    auto idHelmet = (uint8_t) object.getHelmetID();
    auto idHead = (uint8_t) object.getHeadID();
    auto idBody = (uint8_t) object.getBodyID();
    auto idShield = (uint8_t) object.getShieldID();
    auto idWeapon = (uint8_t) object.getWeaponID();
    auto attackBy = (uint8_t) object.getAttackWeapon();
    encodeMsg.push_back(idHelmet);
    encodeMsg.push_back(idHead);
    encodeMsg.push_back(idBody);
    encodeMsg.push_back(idShield);
    encodeMsg.push_back(idWeapon);
    encodeMsg.push_back(ZERO); //Correspondiente al valor del item que no es
    encodeStatePlayer(object, encodeMsg);
    encodeMsg.push_back(attackBy);
}

std::vector<uint8_t> Decoder::encodeGameObjects(const std::vector<GameObjectInfo> &objects) {
    std::vector<uint8_t> encodeMsg;
    uint32_t cantObjects = objects.size();
    uint32_t length = cantObjects * OBJECTLENGTH + 4;
    length = htonl(length);
    cantObjects = htonl(cantObjects);
    uint8_t type = OBJECTSINFOMSG;
    conversorTo8(length, 32, encodeMsg);
    encodeMsg.push_back(type);
    conversorTo8(cantObjects, 32, encodeMsg);
    uint16_t id;
    for(auto& object: objects) {
```

```
        id = htons(object.getId());
        conversorTo8(id,16, encodeMsg);
        if(object.isItem()) {
            encodeItem(object, encodeMsg);
        } else {
            encodeCharacter(object, encodeMsg);
        }
    }
    return encodeMsg;
}

std::vector<GameObjectInfo> Decoder::decodeGameObjects(Message msg) {
    std::vector<GameObjectInfo> objects;
    msg.clear();
    uint32_t cantObjects = ntohl(msg.read32());
    std::string equipment;
    for (uint i=0; i < cantObjects ; i++) {
        uint16_t id = ntohs(msg.read16());
        uint8_t type = msg.read8();
        equipment = decodeEquipment(msg, true);
        auto state = (CharacterStateID) (ntohs(msg.read16()));
        auto dir = (Direction) (ntohs(msg.read16()));
        uint16_t x = ntohs(msg.read16());
        uint16_t y = ntohs(msg.read16());
        auto attackBy = (WeaponID) msg.read8();
        GameObjectInfo info(id,Point(x,y),equipment,dir,state,type,attackBy);
        objects.push_back(info);
    }
    return objects;
}

std::vector<uint8_t> Decoder::encodeCommand(InputInfo input) {
    std::vector<uint8_t> encodeMsg;
    uint32_t length = COMMANDLENGTH;
    length = htonl(length);
    conversorTo8(length,32, encodeMsg);
    uint8_t type = COMMANDMSG;
    encodeMsg.push_back(type);
    auto inputId = (uint8_t) input.input;
    encodeMsg.push_back(inputId);

    uint16_t x = htons(input.position.x);
    conversorTo8(x,16, encodeMsg);
    uint16_t y = htons(input.position.y);
    conversorTo8(y,16, encodeMsg);
    uint16_t adition = htons(input.aditional);
    conversorTo8(adition,16, encodeMsg);

    return encodeMsg;

}

InputInfo Decoder::decodeCommand(Message msg) {
    msg.clear();
    InputInfo input;
    input.input = (InputID) msg.read8();
    input.position.x = ntohs(msg.read16());
    input.position.y = ntohs(msg.read16());
    input.aditional = ntohs(msg.read16());
    return input;
}

std::vector<uint8_t> Decoder::encodeInit(RaceID race, GameClassID gameClass) {
    std::vector<uint8_t> encodeMsg;
    uint32_t length = INITLENGTH;
    conversorTo8(htonl(length),32, encodeMsg);
```



```
    encodeMsg.push_back(INITMSG);
    auto breed = (uint8_t) race;
    auto playerClass = (uint8_t) gameClass;
    encodeMsg.push_back(breed);
    encodeMsg.push_back(playerClass);
    return encodeMsg;
}

std::vector<uint8_t> Decoder::encodeMap(const TiledMap &tiledMap) {
    std::vector<uint8_t> encodeMsg;
    uint32_t length = MAPLENGTH;
    conversorTo8(htonl(length), 32, encodeMsg);
    encodeMsg.push_back(MAPMSG);
    conversorTo8(htons(tiledMap.getWidth()), 16, encodeMsg);
    conversorTo8(htons(tiledMap.getHeight()), 16, encodeMsg);
    encodeMsg.push_back(tiledMap.getTileWidth());
    encodeMsg.push_back(tiledMap.getTileHeight());
    std::vector<TileLayer> tilesLayers = tiledMap.getTileLayers();
    uint8_t amountLayers = tilesLayers.size();
    uint16_t dataSize = tilesLayers.at(0).getData().size();
    encodeMsg.push_back(amountLayers);
    conversorTo8(htons(dataSize), 16, encodeMsg);
    for (auto &aTileLayer : tilesLayers) {
        encodeMsg.push_back(aTileLayer.isGroundLayer() ? 0X01 : ZERO);
        for (auto &aDataTile : aTileLayer.getData()) {
            conversorTo8(htons(aDataTile), 16, encodeMsg);
        }
    }
    std::vector<TileSet> tilesSet = tiledMap.getTilesets();
    uint8_t amountTileSets = tilesSet.size();
    encodeMsg.push_back(amountTileSets);
    for (auto &aTileSet : tilesSet) {
        conversorTo8(htons(aTileSet.getFirstgid()), 16, encodeMsg);
        encodeMsg.push_back(aTileSet.getId());
    }
    return encodeMsg;
}

TiledMap Decoder::decodeMap(Message msg) {
    uint16_t width = ntohs(msg.read16());
    uint16_t height = ntohs(msg.read16());
    uint8_t tileWidth = msg.read8();
    uint8_t tileHeight = msg.read8();
    uint8_t amountLayers = msg.read8();
    uint16_t dataSize = ntohs(msg.read16());
    std::vector<TileLayer> tileLayers;
    for (size_t i = 0; i < amountLayers; ++i) {
        std::vector<uint16_t> data;
        bool isGround = msg.read8();
        for (size_t j = 0; j < dataSize; ++j) {
            data.push_back(ntohs(msg.read16()));
        }
        tileLayers.emplace_back(data, isGround);
    }
    std::vector<TileSet> tileSets;
    uint8_t amountTileSets = msg.read8();
    for (size_t i = 0; i < amountTileSets; ++i) {
        uint16_t firstGid = ntohs(msg.read16());
        uint8_t id = msg.read8();
        tileSets.emplace_back(firstGid, id);
    }
    return TiledMap(width, height, tileWidth, tileHeight, tileLayers, tileSets);
}

std::vector<uint8_t> Decoder::encodeNPCInfo(const NPCInfo& info) {
    std::vector<uint8_t> encodeMsg;
```

jul 21, 20 15:20 **Decoder.cpp** Page 7/8

```
387     uint32_t length = 0;
388     conversorTo8(htonl(length), 32, encodeMsg);
389     encodeMsg.push_back(INTERACTMSG);
390     length += 1;
391     encodeMsg.push_back(info.type);
392     uint8_t cantAccions = info.actions.size();
393     length += 1;
394     encodeMsg.push_back(cantAccions);
395     for (uint j = 0; j< info.actions.size();j++) {
396         length += 1;
397         encodeMsg.push_back(uint8_t(info.actions[j]));
398     }
399     if (info.type ≠ 0) {
400         uint8_t cantItems = info.items.size();
401         length += 1;
402         encodeMsg.push_back(cantItems);
403         for (auto iter: info.items) {
404             length += 3;
405             encodeMsg.push_back(uint8_t(iter.first));
406             conversorTo8(htons(iter.second), 16, encodeMsg);
407         }
408     } else {
409         uint8_t cantItems = ZERO;
410         length += 1;
411         encodeMsg.push_back(cantItems);
412     }
413     uint32_t gold = info.gold;
414     length += 4;
415     conversorTo8(htonl(gold),32,encodeMsg);
416     if (info.type ≡ 3) {
417         uint8_t cantItems = info.itemsInBank.size();
418         length += 1;
419         encodeMsg.push_back(cantItems);
420         for (auto iter: info.itemsInBank) {
421             length += 1;
422             encodeMsg.push_back(uint8_t(iter));
423         }
424     } else {
425         uint8_t cantItems = ZERO;
426         length += 1;
427         encodeMsg.push_back(cantItems);
428     }
429     length = htonl(length);
430     auto* ptr = (uint8_t*) &length;
431     for(int i = 0; i < 4; i++)
432         encodeMsg.at(i) = *(ptr+i);
433
434     return encodeMsg;
435 }
436
437 NPCInfo Decoder::decodeNPCInfo(Message msg){
438     NPCInfo info;
439     info.type = msg.read8();
440     uint8_t cantAccions = msg.read8();
441     for (uint8_t i=0; i < cantAccions; i++) {
442         info.actions.push_back(ActionsProfessionID(msg.read8()));
443     }
444     uint8_t cantItems = msg.read8();
445     for (uint8_t i=0; i < cantItems; i++) {
446         ItemsInventoryID id = ItemsInventoryID(msg.read8());
447         uint16_t cost = ntohs(msg.read16());
448         info.items.insert({id,cost});
449     }
450     info.gold = ntohl(msg.read32());
451     uint8_t cantItemsInBank = msg.read8();
452     for (uint8_t i=0; i < cantItemsInBank; i++) {
```

jul 21, 20 15:20 **Decoder.cpp** Page 8/8

```
453         info.itemsInBank.push_back(ItemsInventoryID(msg.read8()));
454     }
455     return info;
456 }
457
458 Decoder::~Decoder() = default;
```

jul 21, 20 15:20 **DataQueue.h** Page 1/1

```
#ifndef DATAQUEUE_H
#define DATAQUEUE_H

#include "BlockingQueue.h"
#include "Message.h"
#include <utility>
#include <vector>

using DataQueue = BlockingQueue<Message>;

#endif
```

jul 21, 20 15:20 **CommunicationProtocol.h** Page 1/1

```
#ifndef COMMUNICATIONPROTOCOL_H
#define COMMUNICATIONPROTOCOL_H

#include <vector>
#include "PlayerInfo.h"
#include <cstdint>
#include "GameObjectInfo.h"
#include "../client/GameMap.h"
#include "Socket.h"
#include "Message.h"

//Wrapper del socket utilizado para la comunicacion.s
class CommunicationProtocol {
private:
    Socket socket;
public:
    CommunicationProtocol();
    explicit CommunicationProtocol(Socket socket);

    void connect(const char* host, const char* port);

    void send(std::vector<uint8_t> msg) const;

    Message receive() const;

    void stop();

    ~CommunicationProtocol();
};

#endif
```

jul 21, 20 15:20 **CommunicationProtocol.cpp** Page 1/1

```cpp
#include "CommunicationProtocol.h"
#include <arpa/inet.h>
#include <utility>
#include <iostream>

CommunicationProtocol::CommunicationProtocol() : socket() {}

CommunicationProtocol::CommunicationProtocol(Socket socket) :
    socket(std::move(socket)) {}


void CommunicationProtocol::connect(const char* host, const char* port) {
    this→socket.connect(host, port);
}

void CommunicationProtocol::send(std::vector<uint8_t> msg) const {
    this→socket.send(msg.data(), msg.size());
}

Message CommunicationProtocol::receive() const {
    uint8_t length[5];
    this→socket.recieve(&length, 5);
    uint32_t* temp32 = (uint32_t*) length;
  uint32_t length_message = ntohl(*temp32);
    uint8_t type = length[4];

    std::vector<uint8_t> buffer(length_message,0);
    uint8_t* buf = buffer.data();
    this→socket.recieve(buf, length_message);
    return Message(buffer,length_message,type);
}

void CommunicationProtocol::stop() {
    this→socket.shutdown(SHUT_RDWR);
    this→socket.close();
}

CommunicationProtocol::~CommunicationProtocol() = default;
```

jul 21, 20 15:20 **Chrono.h** Page 1/1

```cpp
#ifndef CHRONO_H
#define CHRONO_H
#include <chrono>

class Chrono {
private:
    std::chrono::high_resolution_clock::time_point time;
public:
    Chrono();

    //Devuelve la cantidad de milisegundos que hubo entre 2 llamadas a esta
    //funcion o con la creaciÃ³n del cronÃ³metro.
    double lap();

    ~Chrono();
};

#endif
```

jul 21, 20 15:20 **Chrono.cpp** Page 1/1

```cpp
#include "Chrono.h"

Chrono::Chrono() {
    this→time = std::chrono::high_resolution_clock::now();
}

double Chrono::lap() {
    std::chrono::high_resolution_clock::time_point now =
        std::chrono::high_resolution_clock::now();

    double dt = std::chrono::duration_cast<std::chrono::duration<double>>(now - time).count();
    this→time = now;
    return dt;
}

Chrono::~Chrono() {}
```

jul 21, 20 15:20 **BlockingQueue.h** Page 1/2

```cpp
#ifndef BLOCKING_QUEUE_H
#define BLOCKING_QUEUE_H
#include <mutex>
#include <condition_variable>
#include <queue>
#include <atomic>
#include "Exception.h"

#define ERRORMSG "No hay elementos en la cola y no se bloquea al realizar el pop"
#define CLOSEDQUEUE "La queue se encuentra cerrada"

//La clase Blocking Queue permite un acceso controlado a los recursos
//encolados para que cada uno de los jugadores saquen un numero a la vez.
template<typename ITEM>
class BlockingQueue {
private:
  std::queue<ITEM> blocking_queue;
    mutable std::mutex m;
    std::condition_variable cv;
  bool blockAtPop;
  bool isClosed;

public:
  //Constructores para la clase Blocking Queue, se permite el constructor
  //por movimiento, no asi por copia dado que no nos interesa que una queue
  //sea copiada.
  BlockingQueue(bool blockPop) : blockAtPop(blockPop), isClosed(false){}

  BlockingQueue(BlockingQueue ∧other) {
    std::unique_lock<std::mutex> lock(m);
      this→blocking_queue = std::move(other.blocking_queue);
  }

    BlockingQueue& operator=(BlockingQueue ∧other) {
    std::unique_lock<std::mutex> lock(m);
      this→blocking_queue = std::move(other.blocking_queue);
      return *this;
  }

  //Se realiza el pop del primer elemento de la queue y lo retorna. En caso
  //de que estÃ© vacÃ-a, se queda esperando a que aparezca una nueva unidad
  //del recurso. En caso de que la queue estÃ© cerrada lanza una excepciÃ³n.
  ITEM pop() {
    std::unique_lock<std::mutex> lock(m);
    while(blocking_queue.empty() ∧ blockAtPop){
      if(isClosed) {
        throw Exception(CLOSEDQUEUE);
      }
      cv.wait(lock);
    }
    if (blocking_queue.empty() ∧ ¬blockAtPop)
      throw Exception(ERRORMSG);
    ITEM value(std::move(blocking_queue.front()));
    blocking_queue.pop();
    return value;
  }

  //Se pushea el value en la cola bloqueante. Luego notifica a todos aquellos
  //que estÃ©n esperando.
  void push(ITEM value) {
    std::unique_lock<std::mutex> lock(m);
    blocking_queue.push(std::move(value));
    cv.notify_all();
  }

  bool empty() const {
```

jul 21, 20 15:20 **BlockingQueue.h** Page 2/2

```
67      std::unique_lock<std::mutex> lock(m);
68      return this→blocking_queue.empty();
69    }
70
71    void close() {
72      std::unique_lock<std::mutex> lock(m);
73      isClosed = true;
74      cv.notify_all();
75    }
76
77    ~BlockingQueue(){}
78
79    BlockingQueue(const BlockingQueue&) = delete;
80      BlockingQueue &operator=(const BlockingQueue&) = delete;
81  };
82
83  #endif
```

jul 21, 20 15:20 **Window.h** Page 1/1

```
1   #ifndef WINDOW_H
2   #define WINDOW_H
3
4   class SDL_Window;
5   class SDL_Renderer;
6   union SDL_Event;
7
8   class Window {
9   private:
10    SDL_Window* window;
11    SDL_Renderer* renderer;
12    bool isMinimized;
13    int height;
14    int width;
15    void init();
16    const char* title;
17  public:
18    Window(const char* title);
19    Window(const int height,const int width,const char* title);
20
21    void clearScreen();
22
23    void render();
24
25    void handleEvent(SDL_Event& event);
26
27    SDL_Renderer& getRenderer() const;
28
29    int getWidth() const;
30      int getHeight() const;
31
32    ~Window();
33
34    Window(const Window&) = delete;
35      Window &operator=(const Window&) = delete;
36  };
37
38  #endif
```

jul 21, 20 15:20 **Window.cpp** Page 1/3

```cpp
#include "Window.h"
#include <exception>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_mixer.h>
#include <SDL2/SDL_ttf.h>
#include "../common/Exception.h"

#define CHANNELS 2
#define MIX_CHUNKSIZE 1024
#define WINDOW_HEIGHT 600
#define WINDOW_WIDTH 800

Window::Window(const int height, const int width, const char* title) :
    isMinimized(false), height(height), width(width),title(title) {

  init();

  Uint32 flags = SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE | SDL_RENDERER_ACCELERATED;
  if (SDL_CreateWindowAndRenderer(this→width, this→height, flags, &this→window,
                  &this→renderer))
    throw Exception("Error with SDL_CreateWindowAndRenderer: %s",SDL_GetError());
  SDL_SetWindowTitle(this→window,this→title);
}

Window::Window(const char* title) : isMinimized(false), height(WINDOW_HEIGHT),
  width(WINDOW_WIDTH), title(title) {
    init();

  Uint32 flags = SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE | SDL_RENDERER_ACCELERATED;
  if (SDL_CreateWindowAndRenderer(this→width, this→height, flags, &this→window,
                  &this→renderer))
    throw Exception("Error with SDL_CreateWindowAndRenderer: %s",SDL_GetError());
  SDL_SetWindowTitle(this→window,this→title);
}

void Window::init() {
  if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) < 0)
    throw Exception("Error with SDL_Init: %s",SDL_GetError());

  if (Mix_OpenAudio(MIX_DEFAULT_FREQUENCY, MIX_DEFAULT_FORMAT,
                    CHANNELS, MIX_CHUNKSIZE) ≡ -1) {
      throw Exception("Fail Mix_OpenAudio: %s", Mix_GetError());
  }

  if (¬(IMG_Init(IMG_INIT_PNG) & IMG_INIT_PNG)) {
    throw Exception("Fail IMG_Init: %s", IMG_GetError());
  }

  if(TTF_Init() ≡ -1)
    throw Exception("Fail TTF_Init: %s", TTF_GetError());
}

void Window::clearScreen() {
  SDL_SetRenderDrawColor(this→renderer, 0x00, 0x00, 0x00, 0x00);
  SDL_RenderClear(this→renderer);
}

void Window::handleEvent(SDL_Event& event) {
  if(event.type ≡ SDL_WINDOWEVENT) {
    switch(event.window.event) {
      //Redimenciona el tamaÃ±o de la ventana.
```

jul 21, 20 15:20 **Window.cpp** Page 2/3

```cpp
      case SDL_WINDOWEVENT_SIZE_CHANGED:
        this→width = event.window.data1;
        this→height = event.window.data2;
        render();
        break;
      //Repaint on exposure
      case SDL_WINDOWEVENT_EXPOSED:
        this→isMinimized = false;
        render();
        break;
      //Window minimized
      case SDL_WINDOWEVENT_MINIMIZED:
            this→isMinimized = true;
            break;
      //Window maxized
      case SDL_WINDOWEVENT_MAXIMIZED:
        this→isMinimized = false;
        render();
            break;
      //Window restored
      case SDL_WINDOWEVENT_RESTORED:
        this→isMinimized = false;
        render();
            break;
    }
  }
}

void Window::render() {
  if (¬this→isMinimized) {
    SDL_RenderPresent(this→renderer);
  }
}

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

int Window::getWidth() const {
  int w,h;
  SDL_GL_GetDrawableSize(this→window,&w,&h);
  return w;
}


int Window::getHeight() const {
  int w,h;
  SDL_GL_GetDrawableSize(this→window,&w,&h);
  return h;
}

Window::~Window() {
  if (this→renderer){
    SDL_DestroyRenderer(this→renderer);
    this→renderer = nullptr;
  }
  if (this→window) {
    SDL_DestroyWindow(this→window);
    this→window = nullptr;
  }

  TTF_Quit();
  IMG_Quit();
  int numtimesopened, frequency, channels;
  Uint16 format;
  numtimesopened = Mix_QuerySpec(&frequency, &format, &channels);
```

jul 21, 20 15:20 **Window.cpp** Page 3/3

```
129     for (int i = 0; i < numtimesopened; i++)
130       Mix_CloseAudio();
131     while(Mix_Init(0))
132         Mix_Quit();
133     SDL_Quit();
134 }
```

jul 21, 20 15:20 **UnequipButton.h** Page 1/1

```
1  #ifndef UNEQUIPBUTTON_H
2  #define UNEQUIPBUTTON_H
3
4  #include "RaisedButton.h"
5  #include "../Player.h"
6
7  class UnequipButton : public RaisedButton {
8      Player* player;
9  public:
10     UnequipButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect
   position,
11             const TextureManager& manager,Player* player);
12
13     virtual bool inside(int x, int y);
14
15     virtual void render();
16
17     virtual InputInfo onClick(int item);
18
19     void setViewport(SDL_Rect viewport);
20
21     virtual ~UnequipButton();
22 };
23
24 #endif
```

jul 21, 20 15:20 **UnequipButton.cpp** Page 1/1

```cpp
#include "UnequipButton.h"

UnequipButton::UnequipButton(SDL_Renderer* renderer, Font& font,std::string text
,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo UnequipButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→unequipItem(item);
}

bool UnequipButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void UnequipButton::render() {
    RaisedButton::render();
}

void UnequipButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

UnequipButton::~UnequipButton(){}
```

jul 21, 20 15:20 **UI.h** Page 1/2

```cpp
#ifndef UI_H
#define UI_H

#include <SDL2/SDL.h>
#include "../Window.h"
#include "../../common/Point.h"
#include "../MusicManager.h"
#include "../Player.h"
#include "../Font.h"
#include "SelectButton.h"
#include "RaisedButton.h"
#include "ArrowButton.h"
#include <vector>
#include "NPCInterface.h"
union SDL_Event;

//Clase encargada de renderizar el estado general de todo el jugador.

class UI {
private:
    Player* playerTarget = nullptr; //Jugador del cual debe mostrar estadisticas
.
    Window& window;
    const TextureManager& manager;
    const MusicManager& mixer;
    Font font;
    //Interfaz del NPC con el que se estÃ¡ interactuando
    std::shared_ptr<NPCInterface> npc{nullptr};

    std::vector<SDL_Texture*> texts; //Textos que se muestran en la interfaz
    std::vector<SDL_Texture*> info; //Informacion de los stats del jugador
    std::vector<std::shared_ptr<RaisedButton>> buttonsInventory; //Botones del i
nventario
    std::vector<ItemsInventoryID> itemsID; //Id de los items que estÃ©n disponib
les en el inventario
    std::vector<std::shared_ptr<SelectButton>> buttonsItems; //Items mostrados e
n el inventario
    std::vector<std::shared_ptr<SelectButton>> buttonsBuild; //Items mostrados e
n el equipamiento
    std::shared_ptr<RaisedButton> unequipButton;

    NPCInfo informationNPC;
    int widthSegment;
    int itemSelected{-1};
    int buildSelected{-1};
    int maxExpPreviousLevel{0};
    int maxExpActualLevel{0};

    //Son las funciones encargadas de actualizar el estado general del jugador
    void updateStates();
    void updateHealth();
    void updateMana();
    void updateGold();
    void updateLevelAndExpirience();
    void deleteInfo();

    //Actualiza el inventario del jugador, mostrando por pantalla los items y
    //botones que este tiene disponible para equiparse, vender o tirar.
    void updateInventory();
    void updateItems();

    //Actualiza el equipamiento del jugador en caso de que este no estÃ© interac
tuando
    //En caso de que su estado sea de interactuar, graficarÃ¡ lo que el profesio
nal
    //tenga para mostrar y permitira realizar las acciones disponibles.
```

jul 21, 20 15:20 **UI.h** Page 2/2

```
60      void updateEquipment();
61      void updateInteract();
62      void updateBuild();
63
64      void createTexts();
65
66  public:
67      UI(Window& window, Player* player, const TextureManager& manager, const Musi
    cManager& mixer);
68
69      void render();
70
71      //FunciÃ³n llamada para actualizar la informaciÃ³n del NPC con el que se est
    Ã¡
72      //interactuando.
73      void setNPCInfo(NPCInfo info);
74
75      //Realiza el manejo de los eventos de click.
76      //ActualizarÃ¡ el item seleccionado en el inventario, permitirÃ¡ realizar la
    //acciÃ³n correspondiente a cada botÃ³n o delegarÃ¡ ese manejo al NPCInterfa
    ce
77
78      //correspondiente
79      InputInfo handleClick(SDL_Event& event);
80
81      ~UI();
82  };
83
84  #endif
```

jul 21, 20 15:20 **UI.cpp** Page 1/8

```
1   #include <iostream>
2   #include "UI.h"
3   #include "EquipButton.h"
4   #include "DropButton.h"
5   #include "UnequipButton.h"
6   #include "PriestInterface.h"
7   #include "MerchantInterface.h"
8   #include "BankerInterface.h"
9   #include "../Effect.h"
10
11  #define WIDTHBUTTON 70
12  #define HEIGTHBUTTON 25
13  #define ITEMSMERCHANT 12
14  #define MERCHANTTYPE 1
15  #define PRIESTTYPE 2
16  #define BANKERTYPE 3
17  #define LIMITINVENTORY 9
18  #define TOPBARHEIGHT 60
19
20  UI::UI(Window& window, Player* player, const TextureManager& manager, const Musi
    cManager& mixer) :
21   playerTarget(player),window(window), manager(manager), mixer(mixer),
22   font("assets/font/Prince Valiant.ttf",18,{0xA4, 0xA4, 0xA4}), texts(), info(),
23   itemsID(), buttonsBuild() {
24    createTexts();
25      SDL_Rect buttonRect;
26      for (int i = 0; i< LIMITINVENTORY; i++) {
27          buttonRect = {9+(i%3)*50,50+(i/3)*50,32,32};
28          std::shared_ptr<SelectButton> button = std::shared_ptr<SelectButton>(new
29              SelectButton(&(window.getRenderer()),buttonRect,manager,i));
30          this→buttonsItems.push_back(button);
31      }
32      std::shared_ptr<RaisedButton> equipButton = std::shared_ptr<RaisedButton>(ne
    w EquipButton(&(window.getRenderer()),font,
33          "Equipar",{9,205,WIDTHBUTTON,HEIGTHBUTTON}, manager,playerTarget));
34      std::shared_ptr<RaisedButton> dropButton = std::shared_ptr<RaisedButton>(new
35          DropButton(&(window.getRenderer()),font,"Tirar",{109,205,WIDTHBUTTON,HEIG
    THBUTTON}, manager,playerTarget));
36      this→buttonsInventory.push_back(equipButton);
37      this→buttonsInventory.push_back(dropButton);
38      this→informationNPC.type = 0;
39  }
40
41  void UI::updateHealth(){
42      int w, h,width_text,height_text;
43      SDL_QueryTexture(this→texts[0], NULL, NULL, &w, &h);
44
45      int playerHealth = this→playerTarget→getHealth();
46      int playerMaxHealth = this→playerTarget→getMaxHealth();
47      int health = (playerHealth * (widthSegment-2))/playerMaxHealth;
48      std::string life = "("+ std::to_string(playerHealth) + "/" + std::to_string(
    playerMaxHealth) + ")";
49      SDL_Texture* lifeTexture = font.createText(life,&(window.getRenderer()), &wi
    dth_text, &height_text);
50      this→info.push_back(lifeTexture);
51
52      SDL_Rect healthText = {9+widthSegment*0,10,w,h};
53      SDL_Rect lifeRect = {12+widthSegment*0+w,10,width_text,height_text};
54      SDL_RenderCopy(&(window.getRenderer()), this→texts[0], NULL, &healthText);
55      SDL_RenderCopy(&(window.getRenderer()), this→info[0],NULL,&lifeRect);
56
57      SDL_Rect healthRect = {9+widthSegment*0,30,widthSegment,20};
58      SDL_SetRenderDrawColor(&(this→window.getRenderer()), 0xA4, 0xA4, 0xA4, 0xAA
    );
```

```
59    SDL_RenderDrawRect(&(this→window.getRenderer()), &healthRect);
60
61      SDL_Rect fillHealth = {10+widthSegment*0,31, health,18};
62    SDL_SetRenderDrawColor(&(this→window.getRenderer()), 0xFF, 0x00, 0x00, 0xFF);
63    SDL_RenderFillRect(&(this→window.getRenderer()), &fillHealth);
64  }
65
66  void UI::updateMana(){
67      int w, h,width_text,height_text;
68      SDL_QueryTexture(this→texts[1], NULL, NULL, &w, &h);
69
70      int playerMana = this→playerTarget→getMana();
71      int playerMaxMana = this→playerTarget→getMaxMana();
72      int mana = playerMaxMana > 0 ? (playerMana * (widthSegment-2))/playerMaxMana
     : 0;
73      std::string manaTotal = "("+ std::to_string(playerMana) + "/" + std::to_stri
    ng(playerMaxMana) + ")";
74      SDL_Texture* manaTexture = font.createText(manaTotal,&(window.getRenderer())
    , &width_text, &height_text);
75      this→info.push_back(manaTexture);
76
77      SDL_Rect manaText = {9+widthSegment*2,10,w,h};
78      SDL_Rect manaTotalRect = {12+widthSegment*2+w,10,width_text,height_text};
79      SDL_RenderCopy(&(window.getRenderer()), this→texts[1], NULL, &manaText);
80      SDL_RenderCopy(&(window.getRenderer()), this→info[1],NULL,&manaTotalRect);
81
82      SDL_Rect manaRect = {9+widthSegment*2,30,widthSegment,20};
83      SDL_SetRenderDrawColor(&(this→window.getRenderer()), 0xA4, 0xA4, 0xA4, 0xAA
    );
84    SDL_RenderDrawRect(&(this→window.getRenderer()), &manaRect);
85
86      SDL_Rect fillMana = {10+widthSegment*2,31, mana,18};
87    SDL_SetRenderDrawColor(&(this→window.getRenderer()), 0x01, 0xDF, 0xD7, 0xFF);
88    SDL_RenderFillRect(&(this→window.getRenderer()), &fillMana);
89  }
90
91  void UI::updateGold() {
92      int width_text,height_text;
93      const Texture& goldTexture = manager.getTexture(TextureID::Gold);
94
95      int safeGold = this→playerTarget→getSafeGold();
96      int actualGold = this→playerTarget→getGold();
97      std::string gold = std::to_string(actualGold) + "/" + std::to_string(safeGol
    d);
98      SDL_Texture* goldText = font.createText(gold,&(window.getRenderer()), &width
    _text, &height_text);
99      this→info.push_back(goldText);
100
101     SDL_Rect srcGold = {0,0,32,32};
102     SDL_Rect dstGold = {widthSegment*4,10,32,32};
103     goldTexture.render(srcGold,dstGold);
104
105     SDL_Rect goldRect = {36+widthSegment*4,10,width_text,height_text};
106     SDL_RenderCopy(&(window.getRenderer()), this→info[2],NULL,&goldRect);
107 }
108
109 void UI::updateLevelAndExpirience() {
110     int w, h,width_text,height_text;
111     SDL_QueryTexture(this→texts[2], NULL, NULL, &w, &h);
112
113     const Texture& star = manager.getTexture(TextureID::Star);
114     SDL_Rect srcStar = {0,0,15,16};
115     SDL_Rect dstStar = {20+widthSegment*5,15,15,16};
116     star.render(srcStar,dstStar);
```



```
117
118     std::string level = std::to_string(this→playerTarget→getLevel());
119     SDL_Texture* levelTexture = font.createText(level,&(window.getRenderer()), &
    width_text, &height_text);
120     this→info.push_back(levelTexture);
121
122     SDL_Rect levelText = {50+widthSegment*5,10,w,h};
123     SDL_Rect levelRect = {50+w+widthSegment*5,10,width_text,height_text};
124     SDL_RenderCopy(&(window.getRenderer()), this→texts[2], NULL, &levelText);
125     SDL_RenderCopy(&(window.getRenderer()), this→info[3],NULL,&levelRect);
126
127     int playerExp = this→playerTarget→getExp();
128     int playerMaxExp = this→playerTarget→getMaxExp();
129     int exp = ((playerExp-this→maxExpPreviousLevel) * (widthSegment-2))/(player
    MaxExp-this→maxExpPreviousLevel);
130     std::string expTotal = "("+ std::to_string(playerExp) + "/" + std::to_string
    (playerMaxExp) + ")";
131     SDL_Texture* expTexture = font.createText(expTotal,&(window.getRenderer()),
    &width_text, &height_text);
132     this→info.push_back(expTexture);
133
134     SDL_QueryTexture(this→texts[3], NULL, NULL, &w, &h);
135     SDL_Rect expText = {25+widthSegment*6,8,w,h};
136     SDL_Rect expTotalRect = {25+widthSegment*6+w,8,width_text,height_text};
137     SDL_RenderCopy(&(window.getRenderer()), this→texts[3], NULL, &expText);
138     SDL_RenderCopy(&(window.getRenderer()), this→info[4],NULL,&expTotalRect);
139
140     SDL_Rect expRect = {25+widthSegment*6,30,widthSegment,20};
141     SDL_SetRenderDrawColor(&(this→window.getRenderer()), 0xA4, 0xA4, 0xA4, 0xAA
    );
142   SDL_RenderDrawRect(&(this→window.getRenderer()), &expRect);
143
144     SDL_Rect fillExp = {26+widthSegment*6,31, exp,18};
145   SDL_SetRenderDrawColor(&(this→window.getRenderer()), 0x00, 0xFF, 0x00, 0xFF);
146   SDL_RenderFillRect(&(this→window.getRenderer()), &fillExp);
147
148     if (this→maxExpActualLevel ≠ playerMaxExp) {
149         this→maxExpPreviousLevel = this→maxExpActualLevel;
150         this→maxExpActualLevel = playerMaxExp;
151     }
152
153 }
154
155 void UI::deleteInfo(){
156     for (auto& text: info) {
157         font.deleteText(text);
158         this→info.pop_back();
159     }
160 }
161
162 void UI::updateStates() {
163     const Texture& topBar = manager.getTexture(TextureID::TopBar);
164     SDL_Rect topBarViewport = {0,0,this→window.getWidth(),TOPBARHEIGHT};
165     SDL_RenderSetViewport(&(this→window.getRenderer()), &topBarViewport);
166     SDL_RenderCopy(&(this→window.getRenderer()), topBar.getTexture(), NULL, NUL
    L);
167     deleteInfo();
168     updateHealth();
169     updateMana();
170     updateGold();
171     updateLevelAndExpirience();
172 }
173
174 void UI::updateItems() {
175     itemsID.clear();
```

jul 21, 20 15:20 **UI.cpp** Page 4/8

```
176     int w,h;
177     SDL_QueryTexture(this→texts[4], NULL, NULL, &w, &h);
178     SDL_Rect invText = {15,15,w,h};
179     SDL_RenderCopy(&(window.getRenderer()), this→texts[4], NULL, &invText);
180 
181     std::string items = this→playerTarget→getInventory();
182     std::string item;
183     int idItem;
184 
185     SDL_Rect src = {0,0,52,52};
186     SDL_Rect dst;
187     for (int i = 0; i < LIMITINVENTORY; i++) {
188         item = items.substr(2*i+i,2);
189         idItem = std::stoi(item);
190         const Texture& item = manager.getTexture((ItemsInventoryID)idItem);
191         itemsID.push_back((ItemsInventoryID)idItem);
192 
193         buttonsItems[i]→setViewport({0,TOPBARHEIGHT,widthSegment*2,(this→windo
    w.getHeight()−TOPBARHEIGHT)/2});
194         dst = {35+(i%3)*widthSegment/2,50+(i/3)*((this→window.getHeight()−TOPBA
    RHEIGHT)/2)/5,widthSegment/3,widthSegment/3};
195         buttonsItems[i]→updatePosition(dst);
196         if (idItem > 0) {
197             buttonsItems[i]→render();
198         }
199         item.render(src,dst);
200     }
201 }
202 
203 void UI::updateInventory() {
204     const Texture& statBackground = manager.getTexture(TextureID::StatsBackgroun
    d);
205     SDL_Rect inventoryViewPort = {0,TOPBARHEIGHT,widthSegment*2,(this→window.ge
    tHeight()−TOPBARHEIGHT)/2};
206     SDL_RenderSetViewport(&(this→window.getRenderer()), &inventoryViewPort);
207     SDL_Rect dst = {0,0,widthSegment*2,(this→window.getHeight()−TOPBARHEIGHT)/2
    };
208     SDL_RenderCopy(&(this→window.getRenderer()), statBackground.getTexture(), N
    ULL, &dst);
209     updateItems();
210     int i = 0;
211     for (auto& button: buttonsInventory){
212         button→setViewport({0,TOPBARHEIGHT,widthSegment*2,(this→window.getHeig
    ht()−TOPBARHEIGHT)/2});
213         button→updatePosition({20+(i%2)*widthSegment,int(((this→window.getHeig
    ht()−TOPBARHEIGHT)/2)−HEIGTHBUTTON*2.5),WIDTHBUTTON,HEIGTHBUTTON});
214         button→render();
215         i++;
216     }
217 }
218 
219 void UI::updateBuild() {
220     int w,h;
221     SDL_QueryTexture(this→texts[5], NULL, NULL, &w, &h);
222     SDL_Rect equipment = {15,15,w,h};
223     SDL_RenderCopy(&(window.getRenderer()), this→texts[5], NULL, &equipment);
224 
225     PlayerInfo info = this→playerTarget→getInfo();
226     const Texture& itemBody = manager.getTexture(info.getBodyID());
227     const Texture& itemHead = manager.getTexture(info.getHeadID());
228     const Texture& itemHelmet = manager.getTexture(info.getHelmetID());
229     const Texture& itemWeapon = manager.getTexture(info.getWeaponID());
230     const Texture& itemShield = manager.getTexture(info.getShieldID());
231 
232     if (this→buttonsBuild.size() ≡ 0) {
233         std::shared_ptr<SelectButton> button;
```

jul 21, 20 15:20 **UI.cpp** Page 5/8

```
234         button = std::shared_ptr<SelectButton>(new SelectButton(
235             &(window.getRenderer()),{widthSegment−widthSegment/6,50,widthSegment
    /3,widthSegment/3},manager,0));
236         this→buttonsBuild.push_back(button);
237         button = std::shared_ptr<SelectButton>(new SelectButton(
238             &(window.getRenderer()),{widthSegment−(widthSegment/3)*2,140,widthSe
    gment/3,widthSegment/3},manager,1));
239         this→buttonsBuild.push_back(button);
240         button = std::shared_ptr<SelectButton>(new SelectButton(
241             &(window.getRenderer()),{widthSegment+widthSegment/3,140,widthSegmen
    t/3,widthSegment/3},manager,2));
242         this→buttonsBuild.push_back(button);
243         this→unequipButton = std::shared_ptr<RaisedButton> (new UnequipButton(
244             &(window.getRenderer()),font,"Guardar",{9,205,WIDTHBUTTON,HEIGTHBUTTO
    N}, manager,playerTarget));
245     }
246 
247     for(uint i = 0; i < this→buttonsBuild.size(); i++){
248         buttonsBuild[i]→setViewport({0,(this→window.getHeight())/2,widthSegmen
    t*2,
249                                 (this→window.getHeight())/2});
250         if (i ≡ 0) {
251             buttonsBuild[i]→updatePosition({widthSegment−widthSegment/6,50,widt
    hSegment/3,widthSegment/3});
252         } else if (i ≡ 1) {
253             buttonsBuild[i]→updatePosition({widthSegment−(widthSegment/3)*2,140
    ,widthSegment/3,widthSegment/3});
254         } else if (i ≡ 2) {
255             buttonsBuild[i]→updatePosition({widthSegment+widthSegment/3,140,wid
    thSegment/3,widthSegment/3});
256         }
257         buttonsBuild[i]→render();
258     }
259 
260     SDL_Rect src = {0,0,52,52};
261     itemHelmet.render(src,{widthSegment−widthSegment/6,50,widthSegment/3,widthSe
    gment/3});
262     itemBody.render(src,{widthSegment−widthSegment/3,117+widthSegment/5,(widthSe
    gment/3)*2,(widthSegment/3)*2});
263     if (this→playerTarget→getHealth() ≡ 0) {
264         itemHead.render({0,0,52,52},{widthSegment−widthSegment/6,85+widthSegment
    /6,widthSegment/3,widthSegment/3});
265     } else {
266         itemHead.render({0,0,17,17},{widthSegment−widthSegment/6,85+widthSegment
    /6,widthSegment/3,widthSegment/3});
267     }
268     itemWeapon.render(src,{widthSegment−(widthSegment/3)*2,140,widthSegment/3,wi
    dthSegment/3});
269     itemShield.render(src,{widthSegment+widthSegment/3,140,widthSegment/3,widthS
    egment/3});
270 
271     this→unequipButton→setViewport({0,(this→window.getHeight())/2,widthSegment
    *2,
272                                 (this→window.getHeight())/2});
273     this→unequipButton→updatePosition({20,int(((this→window.getHeight())/2)−HE
    IGTHBUTTON*2.5),WIDTHBUTTON,HEIGTHBUTTON});
274     this→unequipButton→render();
275 
276 }
277 
278 void UI::updateInteract() {
279     if (this→npc ≡ nullptr) {
280         if (this→informationNPC.type ≡ MERCHANTTYPE) {
281             this→npc = std::shared_ptr<NPCInterface>(new MerchantInterface(info
    rmationNPC,&window,manager,playerTarget));
282             const Effect& effect = mixer.getEffect(MusicID::Merchant);
```

```
283             effect.playEffect();
284         } else if (this→informationNPC.type ≡ PRIESTTYPE) {
285             this→npc = std::shared_ptr<NPCInterface>(new PriestInterface(inform
    ationNPC,&window,manager,playerTarget));
286             const Effect& effect = mixer.getEffect(MusicID::Priest);
287             effect.playEffect();
288         } else if (this→informationNPC.type ≡ BANKERTYPE) {
289             this→npc = std::shared_ptr<NPCInterface>(new BankerInterface(inform
    ationNPC,&window,manager,playerTarget));
290             const Effect& effect = mixer.getEffect(MusicID::Banker);
291             effect.playEffect();
292         } else {
293             updateBuild();
294         }
295     } else {
296         npc→update(this→informationNPC);
297     }
298     if (this→npc ≠ nullptr)
299         this→npc→render();
300 }
301
302 void UI::updateEquipment() {
303     const Texture& statBackground = manager.getTexture(TextureID::StatsBackgroun
    d);
304     SDL_Rect equipmentViewPort = {0,(this→window.getHeight())/2,widthSegment*2,
    (this→window.getHeight())/2};
305     SDL_RenderSetViewport(&(this→window.getRenderer()), &equipmentViewPort);
306     SDL_Rect dst = {0,0,widthSegment*2,(this→window.getHeight())/2};
307     SDL_RenderCopy(&(this→window.getRenderer()), statBackground.getTexture(), N
    ULL, &dst);
308     if (this→playerTarget→getState() ≡ CharacterStateID::Interact) {
309         updateInteract();
310         this→buttonsBuild.clear();
311         this→unequipButton = nullptr;
312         this→buildSelected = −1;
313     } else {
314         this→npc = nullptr;
315         this→informationNPC.type = 0;
316         updateBuild();
317     }
318 }
319
320 void UI::render() {
321     this→widthSegment = this→window.getWidth()/8;
322     updateInventory();
323     updateEquipment();
324     updateStates();
325 }
326
327 void UI::setNPCInfo(NPCInfo info) {
328     this→informationNPC = info;
329 }
330
331 InputInfo UI::handleClick(SDL_Event& event) {
332     int x,y;
333     InputInfo info;
334     info.input  = InputID::nothing;
335     bool newItemSelected = false;
336     if (event.type ≡ SDL_MOUSEMOTION ∨ event.type ≡ SDL_MOUSEBUTTONDOWN ∨ event.
    type ≡ SDL_MOUSEBUTTONUP)
337         SDL_GetMouseState(&x, &y);
338     switch(event.type) {
339         case SDL_MOUSEBUTTONDOWN:
340             for(auto& button: buttonsItems){
341                 newItemSelected = button→inside(x,y);
342                 if (newItemSelected) {
```



```
343                     if(itemSelected ≡ −1) {
344                         itemSelected = button→getId();
345                     } else if (itemSelected ≡ button→getId()) {
346                         itemSelected = −1;
347                     } else if (itemSelected ≠ button→getId()) {
348                         buttonsItems[itemSelected]→onClick();
349                         itemSelected = button→getId();
350                     }
351                 }
352             }
353             for (auto& button: buttonsInventory) {
354                 if (button→inside(x,y) ∧ itemSelected ≠ −1)
355                     info = button→onClick(itemSelected+1);
356             }
357             for(auto& button: buttonsBuild){
358                 newItemSelected = button→inside(x,y);
359                 if (newItemSelected) {
360                     if(buildSelected ≡ −1) {
361                         buildSelected = button→getId();
362                     } else if (buildSelected ≡ button→getId()) {
363                         buildSelected = −1;
364                     } else if (buildSelected ≠ button→getId()) {
365                         buttonsBuild[buildSelected]→onClick();
366                         buildSelected = button→getId();
367                     }
368                 }
369             }
370
371             if (this→npc ≠ nullptr) {
372                 int itemId = 0;
373                 if (itemSelected ≠ −1)
374                     itemId = int(itemsID[itemSelected]);
375                 info = this→npc→handleClick(x,y,itemId);
376             } else if (this→unequipButton ≠ nullptr) {
377                 if (this→unequipButton→inside(x,y) ∧ buildSelected > −1)
378                     info = this→unequipButton→onClick(buildSelected);
379             }
380             break;
381
382     }
383     return info;
384 }
385
386 void UI::createTexts() {
387     int width_text, height_text;
388     SDL_Texture* health = font.createText("Vida",
389         &(window.getRenderer()), &width_text, &height_text);
390     this→texts.push_back(health);
391     SDL_Texture* mana = font.createText("Mana",
392         &(window.getRenderer()), &width_text, &height_text);
393     this→texts.push_back(mana);
394     SDL_Texture* level = font.createText("Nivel ",
395         &(window.getRenderer()), &width_text, &height_text);
396     this→texts.push_back(level);
397     SDL_Texture* expirience = font.createText("Experiencia",
398         &(window.getRenderer()), &width_text, &height_text);
399     this→texts.push_back(expirience);
400     SDL_Texture* inventory = font.createText("Inventario",
401         &(window.getRenderer()), &width_text, &height_text);
402     this→texts.push_back(inventory);
403     SDL_Texture* equipment = font.createText("Equipacion",
404         &(window.getRenderer()), &width_text, &height_text);
405     this→texts.push_back(equipment);
406 }
407
408 UI::~UI() {
```

jul 21, 20 15:20 **UI.cpp** Page 8/8

```
409     for (auto& text: texts) {
410         font.deleteText(text);
411     }
412 }
```

jul 21, 20 15:20 **SellButton.h** Page 1/1

```
1  #ifndef SELLBUTTON_H
2  #define SELLBUTTON_H
3
4  #include "RaisedButton.h"
5  #include "../Player.h"
6
7  class SellButton: public RaisedButton {
8  private:
9      Player* player;
10 public:
11     SellButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect pos
   ition,
12          const TextureManager& manager,Player* player);
13
14     virtual bool inside(int x, int y);
15
16     virtual void render();
17
18     virtual InputInfo onClick(int item);
19
20     void setViewport(SDL_Rect viewport);
21
22     virtual ~SellButton();
23 };
24
25 #endif
```

jul 21, 20 15:20 **SellButton.cpp** Page 1/1

```cpp
#include "SellButton.h"

SellButton::SellButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo SellButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→sell(item);
}

bool SellButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void SellButton::render() {
    RaisedButton::render();
}

void SellButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

SellButton::~SellButton(){}
```

jul 21, 20 15:20 **SelectButton.h** Page 1/1

```cpp
#ifndef SELECTBUTTON_H
#define SELECTBUTTON_H

#include "Button.h"

class SelectButton: public Button {
private:
    int id;
public:
    SelectButton(SDL_Renderer* renderer,SDL_Rect position, const TextureManager&
 manager, int id);

    virtual bool inside(int x, int y);

    virtual void render();

    void onClick();

    int getId();

    virtual void updatePosition(SDL_Rect position);

    void setViewport(SDL_Rect viewport);

    ~SelectButton();
};

#endif
```

jul 21, 20 15:20 **SelectButton.cpp** Page 1/1

```cpp
#include "SelectButton.h"

SelectButton::SelectButton(SDL_Renderer* renderer, SDL_Rect position, const TextureManager& manager, int id) :
    Button(renderer,position,manager), id(id){}

void SelectButton::render(){
    int w,h;
    const Texture& background = manager.getTexture(TextureID::Button);
    SDL_QueryTexture(background.getTexture(), NULL, NULL, &w, &h);
    SDL_Rect src = {0,0,w,h};
    background.render(src,this→button);
    if(this→clicked) {
        SDL_Rect click = {button.x−1,button.y−1,button.w+2,button.h+2};
        SDL_SetRenderDrawColor(this→renderer, 0xA4, 0xA4, 0xA4, 0xAA);
      SDL_RenderDrawRect(this→renderer, &click);
    }
    if(this→texture ≠ nullptr){
        SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
        src = {0,0,w,h};
        SDL_Rect dest = {button.x+w/4, button.y,w,h};
        SDL_RenderCopy(this→renderer,this→texture,&src,&dest);
    }
}

void SelectButton::onClick() {
    this→clicked = ¬this→clicked;
}

void SelectButton::updatePosition(SDL_Rect position) {
    Button::updatePosition(position);
}

bool SelectButton::inside(int x, int y) {
    bool inside = true;
    if (x < this→button.x+viewport.x)
        inside = false;
    if (x > this→button.x+this→button.w+viewport.x)
        inside = false;
    if (y < this→button.y+viewport.y)
        inside = false;
    if (y > this→button.y + this→button.h+viewport.y)
        inside = false;
    if (inside)
        onClick();
    return inside;
}

int SelectButton::getId() {
    return this→id;
}

void SelectButton::setViewport(SDL_Rect viewport) {
    this→viewport = viewport;
}

SelectButton::~SelectButton(){}
```

jul 21, 20 15:20 **RetireItemButton.h** Page 1/1

```cpp
#ifndef RETIREITEMBUTTON_H
#define RETIREITEMBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class RetireItemButton: public RaisedButton {
private:
    Player* player;
public:
    RetireItemButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position,
     const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~RetireItemButton();
};

#endif
```

jul 21, 20 15:20 **RetireItemButton.cpp** Page 1/1

```cpp
#include "RetireItemButton.h"

RetireItemButton::RetireItemButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo RetireItemButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→retire(item,true);
}

bool RetireItemButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void RetireItemButton::render() {
    RaisedButton::render();
}

void RetireItemButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

RetireItemButton::~RetireItemButton(){}
```

jul 21, 20 15:20 **RetireGoldButton.h** Page 1/1

```cpp
#ifndef RETIREGOLDBUTTON_H
#define RETIREGOLDBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class RetireGoldButton: public RaisedButton {
private:
    Player* player;
public:
    RetireGoldButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position,
     const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~RetireGoldButton();
};

#endif
```

jul 21, 20 15:20 **RetireGoldButton.cpp** Page 1/1

```cpp
#include "RetireGoldButton.h"

RetireGoldButton::RetireGoldButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo RetireGoldButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→retire(item,false);
}

bool RetireGoldButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void RetireGoldButton::render() {
    RaisedButton::render();
}

void RetireGoldButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

RetireGoldButton::~RetireGoldButton(){}
```

jul 21, 20 15:20 **ResurrectButton.h** Page 1/1

```cpp
#ifndef RESURRECTBUTTON_H
#define RESURRECTBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class ResurrectButton: public RaisedButton {
private:
    Player* player;
public:
    ResurrectButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position,
     const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~ResurrectButton();
};

#endif
```

jul 21, 20 15:20 **ResurrectButton.cpp** Page 1/1

```cpp
#include "ResurrectButton.h"

ResurrectButton::ResurrectButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo ResurrectButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→resurrect();
}

bool ResurrectButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void ResurrectButton::render() {
    RaisedButton::render();
}

void ResurrectButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

ResurrectButton::~ResurrectButton(){}
```

jul 21, 20 15:20 **RaisedButton.h** Page 1/1

```cpp
#ifndef RAISEDBUTTON_H
#define RAISEDBUTTON_H

#include "Button.h"

class RaisedButton : public Button {
public:
    RaisedButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position, const TextureManager& manager);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item) = 0;

    void setViewport(SDL_Rect viewport);

    virtual void updatePosition(SDL_Rect position);

    virtual ~RaisedButton();

};

#endif
```

jul 21, 20 15:20 **RaisedButton.cpp** Page 1/1

```cpp
#include "RaisedButton.h"

RaisedButton::RaisedButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager) :
        Button(renderer,font,text,position,manager) {}


void RaisedButton::render(){
    int w,h;
    const Texture& background = manager.getTexture(TextureID::Button);
    SDL_QueryTexture(background.getTexture(), NULL, NULL, &w, &h);
    SDL_Rect src = {0,0,w,h};
    background.render(src,this→button);
    if(this→clicked) {
        SDL_Rect click = {button.x−1,button.y−1,button.w+2,button.h+2};
        SDL_SetRenderDrawColor(this→renderer, 0xA4, 0xA4, 0xA4, 0xAA);
      SDL_RenderDrawRect(this→renderer, &click);
        this→clicked = ¬this→clicked;
    }
    if(this→texture ≠ nullptr){
        SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
        src = {0,0,w,h};
        SDL_Rect dest = {button.x+(w/6), button.y,w,h};
        SDL_RenderCopy(this→renderer,this→texture,&src,&dest);
    }
}

void RaisedButton::updatePosition(SDL_Rect position) {
    Button::updatePosition(position);
}

bool RaisedButton::inside(int x, int y) {
    bool inside = true;
    if (x < this→button.x+viewport.x)
        inside = false;
    if (x > this→button.x+this→button.w+viewport.x)
        inside = false;
    if (y < this→button.y+viewport.y)
        inside = false;
    if (y > this→button.y + this→button.h+viewport.y)
        inside = false;
    return inside;
}

void RaisedButton::setViewport(SDL_Rect viewport){
    this→viewport = viewport;
}

RaisedButton::~RaisedButton(){}
```

jul 21, 20 15:20 **PriestInterface.h** Page 1/1

```cpp
#ifndef PRIESTINTERFACE_H
#define PRIESTINTERFACE_H

#include "NPCInterface.h"
#include <vector>
#include <memory>
#include "RaisedButton.h"
#include "SelectButton.h"

class PriestInterface: public NPCInterface {
private:
    std::vector<std::shared_ptr<RaisedButton>> buttonsNPC; //Botones que nos mue
stra el NPC
    std::vector<std::shared_ptr<SelectButton>> buttonsItemsNPC; //Items que nos
muestra el NPC
    std::vector<SDL_Texture*> gold; //El oro que se salen cada uno de los items
    std::vector<ItemsInventoryID> itemsPriest;
    int itemSelectedNPC{−1};

    void deleteGoldValues();
public:
    explicit PriestInterface(NPCInfo info,Window* window, const TextureManager&
manager,Player* player);

    virtual void render();

    virtual InputInfo handleClick(int x, int y, int itemSelected);

    ~PriestInterface();

};


#endif
```

```
#include "PriestInterface.h"

#include <memory>

#define WIDTHBUTTON 70
#define HEIGTHBUTTON 25

PriestInterface::PriestInterface(NPCInfo info,Window* window, const TextureManager& manager,Player* player) :
    NPCInterface(info,window,manager,player),buttonsNPC(), buttonsItemsNPC(), gold() {
    int width_text, height_text;
    SDL_Texture* priest = font.createText("Sacerdote",
        &(window→getRenderer()), &width_text, &height_text);
    this→texture = priest;
}

void PriestInterface::render() {
    int w,h;
    int i = 0;
    int width = 0;
    int height = 0;
    SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
    SDL_Rect priest = {15,15,w,h};
    SDL_RenderCopy(&(window→getRenderer()), this→texture, NULL, &priest);

    SDL_Rect src;
    SDL_Rect dst;
    std::shared_ptr<RaisedButton> button;
    std::shared_ptr<SelectButton> selection;
    SDL_Texture* textureGold;
    bool loadButtons = false;

    std::vector<std::pair<ItemsInventoryID, uint>> items(information.items.begin(),information.items.end());

    if (this→itemsPriest.size() ≡ 0) {
        for (auto iter: items) {
            this→itemsPriest.push_back(iter.first);
        }
    }

    if (this→buttonsItemsNPC.size() ≡ 0)
        loadButtons = true;

    int widthSegment = this→window→getWidth()/8;

    for (uint j = 0; j < items.size(); j++) {
        const Texture& item = manager.getTexture(items[j].first);
        src = {0,0,52,52};
        dst = {20+(i%3)*widthSegment/2,50+(i/3)*((this→window→getHeight())/2)/5,widthSegment/3,widthSegment/3};
        if (loadButtons) {
            selection = std::make_shared<SelectButton>(&(window→getRenderer()),dst,manager,j);
            this→buttonsItemsNPC.push_back(selection);
        }
        this→buttonsItemsNPC[i]→setViewport({0,(this→window→getHeight())/2,(this→window→getWidth()/8)*2,(this→window→getHeight())/2});
        this→buttonsItemsNPC[i]→updatePosition({12+width,((this→window→getHeight()/2)-height-HEIGTHBUTTON,WIDTHBUTTON,HEIGTHBUTTON});
        this→buttonsItemsNPC[i]→render();
        item.render(src,dst);
        textureGold = font.createText(std::to_string(items[j].second),&(window→getRenderer()),&w,&h);
        this→gold.push_back(textureGold);
        src = {0,0,w,h};
```



```
        dst = {widthSegment/4+(i%3)*widthSegment/2,15+(i/3)*((this→window→getHeight())/2)/5+widthSegment/2,w,h};
        SDL_RenderCopy(&(window→getRenderer()),textureGold,&src,&dst);
        i++;
    }
    i = 0;
    loadButtons = false;
    if (this→buttonsNPC.size() ≡ 0)
        loadButtons = true;
    for (auto& action: this→information.actions) {
        if (i % 2 ≡ 0) {
            width = 0;
        } else {
            width = 100;
        }
        if (i < this→information.actions.size()/2.0) {
            height = 0;
        } else {
            height = 50;
        }
        if (loadButtons){
            button = createButtonAction(action,{12+width,((this→window→getHeight()-60)/2)-height-HEIGTHBUTTON,WIDTHBUTTON,HEIGTHBUTTON});
            this→buttonsNPC.push_back(button);
        }
        this→buttonsNPC[i]→setViewport({0,(this→window→getHeight())/2,(this→window→getWidth()/8)*2,(this→window→getHeight())/2});
        this→buttonsNPC[i]→updatePosition({12+width,((this→window→getHeight()-60)/2)-height-HEIGTHBUTTON,WIDTHBUTTON,HEIGTHBUTTON});
        this→buttonsNPC[i]→render();
        i++;
    }
}

InputInfo PriestInterface::handleClick(int x, int y, int itemSelected) {
    InputInfo info;
    info.input  = InputID::nothing;
    bool newItemSelected = false;
    for(auto& button: buttonsItemsNPC){
        newItemSelected = button→inside(x,y);
        if (newItemSelected) {
            if(itemSelectedNPC ≡ -1) {
                itemSelectedNPC = button→getId() ;
            } else if (itemSelectedNPC ≡ button→getId()) {
              itemSelectedNPC = -1;
            } else if (itemSelectedNPC ≠ button→getId()) {
                buttonsItemsNPC[itemSelectedNPC]→onClick();
                itemSelectedNPC = button→getId();
            }
        }
    }
    for (uint i = 0; i < this→buttonsNPC.size(); i++) {
        if (buttonsNPC[i]→inside(x,y)) {
            if (i ≡ 2 ∧ itemSelectedNPC ≡ -1)
                continue;
            info = buttonsNPC[i]→onClick(int(itemsPriest[itemSelectedNPC]));
        }
    }
    return info;
}

void PriestInterface::deleteGoldValues() {
    std::vector<SDL_Texture*>::iterator iter;
    iter = this→gold.begin();
    while (iter ≠ this→gold.end()) {
        SDL_DestroyTexture(*iter);
```

jul 21, 20 15:20 **PriestInterface.cpp** Page 3/3

```
122             iter  = gold.erase(iter);
123         }
124     }
125
126
127     PriestInterface::~PriestInterface() {
128         deleteGoldValues();
129     }
```

jul 21, 20 15:20 **NPCInterface.h** Page 1/1

```
1   #ifndef NPCINTERFACE_H
2   #define NPCINTERFACE_H
3
4   #include "../../common/Identificators.h"
5   #include <SDL2/SDL.h>
6   #include "../Window.h"
7   #include "../TextureManager.h"
8   #include <memory>
9   #include "../Player.h"
10  #include "../Font.h"
11  #include "RaisedButton.h"
12
13  class NPCInterface {
14  protected:
15      NPCInfo information;
16      SDL_Texture* texture = nullptr;
17      Window* window;
18      const TextureManager& manager;
19      Player* player;
20      Font font;
21      std::shared_ptr<RaisedButton> createButtonAction(ActionsProfessionID action,
    SDL_Rect rect);
22  public:
23      NPCInterface(NPCInfo info,Window* window, const TextureManager& manager,Play
    er* player);
24
25      virtual void render() = 0;
26
27      void update(NPCInfo npcInfo);
28
29      virtual InputInfo handleClick(int x, int y, int itemSelected) = 0;
30
31      ~NPCInterface();
32
33  };
34
35
36  #endif
```

jul 21, 20 15:20 **NPCInterface.cpp** Page 1/2

```cpp
#include "NPCInterface.h"
#include "SellButton.h"
#include "ResurrectButton.h"
#include "BuyButton.h"
#include "CureButton.h"
#include "RetireGoldButton.h"
#include "DepositGoldButton.h"
#include "DepositItemButton.h"
#include "RetireItemButton.h"

NPCInterface::NPCInterface(NPCInfo info,Window* window, const TextureManager& manager,Player* player)
     : information(info), window(window), manager(manager), player(player),
     font("assets/font/Prince Valiant.ttf",18,{0xA4, 0xA4, 0xA4}){}

std::shared_ptr<RaisedButton> NPCInterface::createButtonAction(ActionsProfessionID action,SDL_Rect rect) {
    std::string text;
    std::shared_ptr<RaisedButton> button;
    switch (action) {
    case ActionsProfessionID::Buy:
        text = "Comprar";
        button = std::shared_ptr<RaisedButton>(new BuyButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::Sell:
        text = "Vender";
        button = std::shared_ptr<RaisedButton>(new SellButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::Resurrect:
        text = "Resucitar";
        button = std::shared_ptr<RaisedButton>(new ResurrectButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::Cure:
        text = "Curar";
        button = std::shared_ptr<RaisedButton>(new CureButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::DepositGold:
        text ="Depositar";
        button = std::shared_ptr<RaisedButton>(new DepositGoldButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::DepositItem:
        text = "Depositar";
        button = std::shared_ptr<RaisedButton>(new DepositItemButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::RetireItem:
        text = "Retirar";
        button = std::shared_ptr<RaisedButton>(new RetireItemButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::RetireGold:
        text = "Retirar";
        button = std::shared_ptr<RaisedButton>(new RetireGoldButton(&(window→getRenderer()),font,text,rect, manager,player));
        break;
    case ActionsProfessionID::Nothing:
        text = "";
        button = nullptr;
        break;
    }
    return button;
```

jul 21, 20 15:20 **NPCInterface.cpp** Page 2/2

```cpp
}


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

void NPCInterface::update(NPCInfo npcInfo) {
    information = npcInfo;
}
```

jul 21, 20 15:20 **MerchantInterface.h** Page 1/1

```h
#ifndef MERCHANTINTERFACE_H
#define MERCHANTINTERFACE_H

#include "NPCInterface.h"
#include <vector>
#include <memory>
#include "RaisedButton.h"
#include "SelectButton.h"
#include "ArrowButton.h"

class MerchantInterface: public NPCInterface {
private:
    std::vector<std::shared_ptr<RaisedButton>> buttonsNPC; //Botones que nos muestra el NPC
    std::vector<std::shared_ptr<SelectButton>> buttonsItemsNPC; //Items que nos muestra el NPC
    std::vector<SDL_Texture*> gold; //El oro que se salen cada uno de los items
    std::vector<ItemsInventoryID> itemsMerchant;
    int itemSelectedNPC{-1};
    uint pagItems{0}; //La pÃ¡gina que nos estÃ¡ mostrando el NPC de los items
    uint pagMax{0}; //La cantidad de pÃ¡ginas que se va a tener con el NPC
    std::shared_ptr<ArrowButton> arrow{nullptr};

    void deleteGoldValues();
public:
    explicit MerchantInterface(NPCInfo info,Window* window, const TextureManager& manager,Player* player);

    virtual void render();

    virtual InputInfo handleClick(int x, int y, int itemSelected);

    ~MerchantInterface();

};


#endif
```

jul 21, 20 15:20 **MerchantInterface.cpp** Page 1/3

```cpp
#include "MerchantInterface.h"

#define WIDTHBUTTON 70
#define HEIGTHBUTTON 25
#define ITEMSMERCHANT 9

MerchantInterface::MerchantInterface(NPCInfo info,Window* window, const TextureManager& manager,Player* player) :
    NPCInterface(info,window,manager,player), buttonsNPC(), buttonsItemsNPC(), gold(), itemsMerchant() {
    int width_text, height_text;
    SDL_Texture* merchant = font.createText("Comerciante",
        &(window→getRenderer()), &width_text, &height_text);
    this→texture = merchant;
}

void MerchantInterface::render() {
    int w,h;
    int i = 0;
    int width = 0;
    SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
    SDL_Rect merchant = {15,15,w,h};
    SDL_RenderCopy(&(window→getRenderer()), this→texture, NULL, &merchant);

    SDL_Rect src;
    SDL_Rect dst;
    std::shared_ptr<RaisedButton> button;
    std::shared_ptr<SelectButton> selection;
    SDL_Texture* textureGold;
    bool loadButtons = false;

    this→pagMax = information.items.size() / ITEMSMERCHANT + 1;
    std::vector<std::pair<ItemsInventoryID, uint>> items(information.items.begin(),information.items.end());

    if (this→itemsMerchant.size() ≡ 0) {
        for (auto iter: items) {
            this→itemsMerchant.push_back(iter.first);
        }
    }

    if (this→buttonsItemsNPC.size() ≡ 0)
        loadButtons = true;

    uint max;
    if ((pagItems+1)*ITEMSMERCHANT > information.items.size()){
        max = information.items.size();
    } else {
        max =(pagItems+1)*ITEMSMERCHANT;
    }
    int widthSegment = this→window→getWidth()/8;
    for (uint j = pagItems*ITEMSMERCHANT; j < max; j++) {
        const Texture& item = manager.getTexture(items[j].first);
        src = {0,0,52,52};
        dst = {20+(i%3)*widthSegment/2,50+(i/3)*((this→window→getHeight())/2)/5,widthSegment/3,widthSegment/3};
        if (loadButtons) {
            selection = std::shared_ptr<SelectButton>(new
            SelectButton(&(window→getRenderer()),dst,manager,j));
            this→buttonsItemsNPC.push_back(selection);
        }
        this→buttonsItemsNPC[i]→setViewport({0,(this→window→getHeight())/2,
                                (this→window→getWidth()/8)*2,(this→window→getHeight())/2});
        this→buttonsItemsNPC[i]→updatePosition(dst);
        this→buttonsItemsNPC[i]→render();
```

jul 21, 20 15:20 **MerchantInterface.cpp** Page 2/3

```
62          item.render(src,dst);
63          textureGold = font.createText(std::to_string(items[j].second),&(window→
    getRenderer()),&w,&h);
64          this→gold.push_back(textureGold);
65          src = {0,0,w,h};
66          dst = {widthSegment/4+(i%3)*widthSegment/2,15+(i/3)*((this→window→getH
    eight())/2)/5+widthSegment/2,w,h};
67          SDL_RenderCopy(&(window→getRenderer()),textureGold,&src,&dst);
68          i++;
69          if (i ≡ ITEMSMERCHANT)
70              break;
71      }
72      i = 0;
73
74      loadButtons = false;
75      if (this→buttonsNPC.size() ≡ 0)
76          loadButtons = true;
77      for (auto& action: this→information.actions) {
78          if (i ≡ 0) {
79              width = 0;
80          } else {
81              width = 100;
82          }
83          if (loadButtons){
84              button = createButtonAction(action,{12+width,
85                      ((this→window→getHeight()−60)/2)−HEIGTHBUTTON,WIDTHBUTTON,
    HEIGTHBUTTON});
86              this→buttonsNPC.push_back(button);
87          }
88          this→buttonsNPC[i]→setViewport({0,(this→window→getHeight())/2,
89                          (this→window→getWidth()/8)*2,(this→window→getHeight()
    )/2});
90          this→buttonsNPC[i]→updatePosition({12+width,
91                      ((this→window→getHeight()−60)/2)−HEIGTHBUTTON,WIDTHBUTTON,
    HEIGTHBUTTON});
92          this→buttonsNPC[i]→render();
93          i++;
94      }
95      if(loadButtons) {
96          SDL_Rect rect = {(this→window→getWidth()/8)*2−40,((this→window→getHei
    ght()−60)/2)−HEIGTHBUTTON*4,20,20};
97          this→arrow = std::shared_ptr<ArrowButton>(new ArrowButton(&(window→get
    Renderer()),font,"+",rect, manager));
98      }
99      if (this→arrow ≠ nullptr){
100         this→arrow→setViewport({0,(this→window→getHeight())/2,
101                     (this→window→getWidth()/8)*2,(this→window→getHeight())/2}
    );
102         this→arrow→updatePosition({(this→window→getWidth()/8)*2−40,
103             ((this→window→getHeight()−60)/2)−HEIGTHBUTTON*4,20,20});
104         this→arrow→render();
105     }
106 }
107
108 InputInfo MerchantInterface::handleClick(int x, int y, int itemSelected) {
109     InputInfo info;
110     info.input  = InputID::nothing;
111     bool newItemSelected = false;
112     for(auto& button: buttonsItemsNPC){
113         newItemSelected = button→inside(x,y);
114         if (newItemSelected) {
115             if(itemSelectedNPC ≡ −1) {
116                 itemSelectedNPC = button→getId() ;
117             } else if (itemSelectedNPC ≡ button→getId()) {
118               itemSelectedNPC = −1;
119             } else if (itemSelectedNPC ≠ button→getId()) {
```

jul 21, 20 15:20 **MerchantInterface.cpp** Page 3/3

```
120                 buttonsItemsNPC[itemSelectedNPC]→onClick();
121                 itemSelectedNPC = button→getId();
122             }
123         }
124     }
125     if (buttonsNPC[0]→inside(x,y) ∧ itemSelectedNPC > −1)
126         info = buttonsNPC[0]→onClick(int(itemsMerchant[itemSelectedNPC+pagItems
    *ITEMSMERCHANT]));
127     if (buttonsNPC[1]→inside(x,y))
128         info = buttonsNPC[1]→onClick(itemSelected);
129
130     if (this→arrow ≠ nullptr ∧ this→arrow→inside(x,y)) {
131         this→arrow→onClick();
132         this→pagItems = (pagItems+1) % this→pagMax;
133     }
134     return info;
135 }
136
137 void MerchantInterface::deleteGoldValues() {
138     std::vector<SDL_Texture*>::iterator iter;
139     iter = this→gold.begin();
140     while (iter ≠ this→gold.end()) {
141         SDL_DestroyTexture(*iter);
142         iter  = gold.erase(iter);
143     }
144 }
145
146
147 MerchantInterface::~MerchantInterface() {
148     deleteGoldValues();
149 }
```

jul 21, 20 15:20 **EquipButton.h** Page 1/1

```
#ifndef EQUIPBUTTON_H
#define EQUIPBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class EquipButton : public RaisedButton {
    Player* player;
public:
    EquipButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect po
sition,
          const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~EquipButton();
};

#endif
```

jul 21, 20 15:20 **EquipButton.cpp** Page 1/1

```
#include "EquipButton.h"

EquipButton::EquipButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo EquipButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→selectItem(item);
}

bool EquipButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void EquipButton::render() {
    RaisedButton::render();
}

void EquipButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

EquipButton::~EquipButton(){}
```

jul 21, 20 15:20 **DropButton.h** Page 1/1

```
#ifndef DROPBUTTON_H
#define DROPBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class DropButton : public RaisedButton {
private:
    Player* player;
public:
    DropButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect pos
ition,
     const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~DropButton();
};

#endif
```

jul 21, 20 15:20 **DropButton.cpp** Page 1/1

```
#include "DropButton.h"

DropButton::DropButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo DropButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→dropItem(item);
}

bool DropButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void DropButton::render() {
    RaisedButton::render();
}

void DropButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

DropButton::~DropButton(){}
```

jul 21, 20 15:20 **DepositItemButton.h** Page 1/1

```
#ifndef DEPOSITITEMBUTTON_H
#define DEPOSITITEMBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class DepositItemButton: public RaisedButton {
private:
    Player* player;
public:
    DepositItemButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position,
     const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~DepositItemButton();
};

#endif
```

jul 21, 20 15:20 **DepositItemButton.cpp** Page 1/1

```
#include "DepositItemButton.h"

DepositItemButton::DepositItemButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo DepositItemButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→deposit(item, true);
}

bool DepositItemButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void DepositItemButton::render() {
    RaisedButton::render();
}

void DepositItemButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

DepositItemButton::~DepositItemButton(){}
```

jul 21, 20 15:20 **DepositGoldButton.h** Page 1/1

```
#ifndef DEPOSITGOLDBUTTON_H
#define DEPOSITGOLDBUTTON_H

#include "RaisedButton.h"
#include "../Player.h"

class DepositGoldButton: public RaisedButton {
private:
    Player* player;
public:
    DepositGoldButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position,
     const TextureManager& manager,Player* player);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual InputInfo onClick(int item);

    void setViewport(SDL_Rect viewport);

    virtual ~DepositGoldButton();
};

#endif
```

jul 21, 20 15:20 **DepositGoldButton.cpp** Page 1/1

```
#include "DepositGoldButton.h"

DepositGoldButton::DepositGoldButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager,Player* player) :
        RaisedButton(renderer,font,text,position,manager), player(player) {}

InputInfo DepositGoldButton::onClick(int item) {
    this→clicked = ¬this→clicked;
    return this→player→deposit(item,false);
}

bool DepositGoldButton::inside(int x, int y) {
    return RaisedButton::inside(x,y);
}

void DepositGoldButton::render() {
    RaisedButton::render();
}

void DepositGoldButton::setViewport(SDL_Rect viewport){
    RaisedButton::setViewport(viewport);
}

DepositGoldButton::~DepositGoldButton(){}
```

jul 21, 20 15:20 **CureButton.h** Page 1/1

```
1  #ifndef CUREBUTTON_H
2  #define CUREBUTTON_H
3
4  #include "RaisedButton.h"
5  #include "../Player.h"
6
7  class CureButton: public RaisedButton {
8  private:
9      Player* player;
10 public:
11     CureButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect pos
   ition,
12      const TextureManager& manager,Player* player);
13
14     virtual bool inside(int x, int y);
15
16     virtual void render();
17
18     virtual InputInfo onClick(int item);
19
20     void setViewport(SDL_Rect viewport);
21
22     virtual ~CureButton();
23 };
24
25 #endif
```

jul 21, 20 15:20 **CureButton.cpp** Page 1/1

```
1  #include "CureButton.h"
2
3  CureButton::CureButton(SDL_Renderer* renderer, Font& font,std::string text,
4      SDL_Rect position, const TextureManager& manager,Player* player) :
5          RaisedButton(renderer,font,text,position,manager), player(player) {}
6
7  InputInfo CureButton::onClick(int item) {
8      this→clicked = ¬this→clicked;
9      return this→player→cure();
10 }
11
12 bool CureButton::inside(int x, int y) {
13     return RaisedButton::inside(x,y);
14 }
15
16 void CureButton::render() {
17     RaisedButton::render();
18 }
19
20 void CureButton::setViewport(SDL_Rect viewport){
21     RaisedButton::setViewport(viewport);
22 }
23
24 CureButton::~CureButton(){}
```

jul 21, 20 15:20 **BuyButton.h** Page 1/1

```
1  #ifndef BUYBUTTON_H
2  #define BUYBUTTON_H
3
4  #include "RaisedButton.h"
5  #include "../Player.h"
6
7  class BuyButton: public RaisedButton {
8  private:
9      Player* player;
10 public:
11     BuyButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect posi
   tion,
12          const TextureManager& manager,Player* player);
13
14     virtual bool inside(int x, int y);
15
16     virtual void render();
17
18     virtual InputInfo onClick(int item);
19
20     void setViewport(SDL_Rect viewport);
21
22     virtual ~BuyButton();
23 };
24
25 #endif
```

jul 21, 20 15:20 **BuyButton.cpp** Page 1/1

```
1  #include "BuyButton.h"
2
3  BuyButton::BuyButton(SDL_Renderer* renderer, Font& font,std::string text,
4      SDL_Rect position, const TextureManager& manager,Player* player) :
5          RaisedButton(renderer,font,text,position,manager), player(player) {}
6
7  InputInfo BuyButton::onClick(int item) {
8      this→clicked = ¬this→clicked;
9      return this→player→buy(item);
10 }
11
12 bool BuyButton::inside(int x, int y) {
13     return RaisedButton::inside(x,y);
14 }
15
16 void BuyButton::render() {
17     RaisedButton::render();
18 }
19
20 void BuyButton::setViewport(SDL_Rect viewport){
21     RaisedButton::setViewport(viewport);
22 }
23
24 BuyButton::~BuyButton(){}
```

jul 21, 20 15:20 **Button.h** Page 1/1

```h
#ifndef BUTTON_H
#define BUTTON_H

#include "../Font.h"
#include "../TextureManager.h"

class Button {
protected:
    std::string text;
    SDL_Rect button;
    int textSize{8};
    SDL_Texture* texture = nullptr;
    const TextureManager& manager;
    SDL_Renderer* renderer;
    SDL_Rect viewport{0,0,0,0};
    bool clicked;

public:
    Button(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position, const TextureManager& manager);

    Button(SDL_Renderer* renderer,SDL_Rect position, const TextureManager& manager);

    virtual bool inside(int x, int y) = 0;

    virtual void render() = 0;

    void onClick();

    virtual void updatePosition(SDL_Rect position);

    void setViewport(SDL_Rect viewport);

    ~Button();
};

#endif
```

jul 21, 20 15:20 **Button.cpp** Page 1/1

```cpp
#include "Button.h"

Button::Button(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect position,
    const TextureManager& manager) : text(text), button(position), manager(manager),
    renderer(renderer), clicked(false) {
    int w,h;
    font.setSize(this→textSize);
    this→texture = font.createText(text,renderer,&w,&h);
}

Button::Button(SDL_Renderer* renderer,SDL_Rect position, const TextureManager& manager) :
    button(position), manager(manager),  renderer(renderer), clicked(false){}

void Button::onClick() {
        this→clicked = ¬this→clicked;
}

void Button::updatePosition(SDL_Rect position) {
    this→button = position;
}

void Button::setViewport(SDL_Rect viewport) {
        this→viewport = viewport;
}

Button::~Button() {
        SDL_DestroyTexture(this→texture);
}
```

jul 21, 20 15:20 **BankerInterface.h** Page 1/1

```h
#ifndef BANKERINTERFACE_H
#define BANKERINTERFACE_H

#include "NPCInterface.h"
#include <vector>
#include <memory>
#include "RaisedButton.h"
#include "SelectButton.h"
#include "ArrowButton.h"

class BankerInterface: public NPCInterface {
private:
    std::vector<std::shared_ptr<RaisedButton>> buttonsNPC; //Botones que nos muestra el NPC
    std::vector<std::shared_ptr<SelectButton>> buttonsItemsNPC; //Items que nos muestra el NPC
    std::shared_ptr<ArrowButton> changeScreen{nullptr};
    std::vector<std::shared_ptr<ArrowButton>> arrows;
    bool pageGold{true};
    int amountGold{0};
    SDL_Texture* goldValue{nullptr};
    SDL_Texture* inBank{nullptr};
    SDL_Texture* goldInBank{nullptr};
    int itemSelectedNPC{-1};
    uint pagItemsInBank{0}; //La pÃ¡gina que nos estÃ¡ mostrando el NPC de los items depositados
    uint pagMaxInBank{0}; //La cantidad de pÃ¡ginas que se va a tener con el NPC
    std::shared_ptr<ArrowButton> arrow{nullptr};

    void renderGoldManagment();
    void renderItems();
public:
    explicit BankerInterface(NPCInfo info,Window* window, const TextureManager& manager,Player* player);

    virtual void render();

    virtual InputInfo handleClick(int x, int y, int itemSelected);

    ~BankerInterface();

};

#endif
```

jul 21, 20 15:20 **BankerInterface.cpp** Page 1/5

```cpp
#include "BankerInterface.h"
#include <SDL2/SDL.h>

#include <memory>
#include <utility>

#define WIDTHBUTTON 70
#define HEIGTHBUTTON 25
#define ITEMSBANKER 9

BankerInterface::BankerInterface(NPCInfo info, Window* window, const TextureManager& manager, Player* player) :
    NPCInterface(std::move(info), window, manager, player), buttonsNPC(), buttonsItemsNPC(), arrows() {
    int width_text, height_text;
    SDL_Texture* banker = font.createText("Banquero",
        &(window->getRenderer()), &width_text, &height_text);
    this->texture = banker;
    this->inBank = font.createText("Oro en Banco: ",
        &(window->getRenderer()), &width_text, &height_text);
    SDL_Rect rect = {(this->window->getWidth()/8)*2-40,((this->window->getHeight()-60)/2)-HEIGTHBUTTON*3,20,20};
    this->changeScreen = std::make_shared<ArrowButton>(&(window->getRenderer()),font,">",rect, manager);
}

void BankerInterface::renderGoldManagment() {
    int w,h;
    int i =0;
    SDL_QueryTexture(this->inBank, NULL, NULL, &w, &h);
    SDL_Rect bank = {15, 45, w, h};
    SDL_RenderCopy(&(window->getRenderer()), this->inBank, NULL, &bank);
    bank = {15, 45 + h, w, h};
    this->goldInBank = font.createText(std::to_string(information.gold),
        &(window->getRenderer()), &w, &h);
    bank.w = w;
    bank.h = h;
    SDL_RenderCopy(&(window->getRenderer()), this->goldInBank, NULL, &bank);

    const Texture& goldTexture = manager.getTexture(TextureID::Gold);
    SDL_Rect srcGold = {0,0,32,32};
    SDL_Rect dstGold = {25,140,32,32};
    goldTexture.render(srcGold,dstGold);
    std::string inputText;
    if (this->amountGold == 0) {
        int amount = this->player->getGold() - this->player->getSafeGold();
        if (amount < 0)
            amount = 0;
        inputText = std::to_string(amount);
    }
    inputText = std::to_string(this->amountGold);
    this->goldValue = font.createText(inputText,
        &(window->getRenderer()), &w, &h);
    SDL_Rect gold = {100,140,w,h};
    SDL_RenderCopy(&(window->getRenderer()), this->goldValue, NULL, &gold);

    bool loadButtons =false;

    if(this->arrows.empty()) {
        SDL_Rect rect = {100,120,20,20};
        this->arrows.push_back(std::make_shared<ArrowButton>(&(window->getRenderer()),font,"+",rect, manager));
        rect = {100,160,20,20};
        this->arrows.push_back(std::make_shared<ArrowButton>(&(window->getRenderer()),font,"-",rect, manager));
    }
```

```
61      for (auto& button: arrows) {
62          button→setViewport({0,(this→window→getHeight())/2,
63                              (this→window→getWidth()/8)*2,(this→window→getHeight())/2}
    );
64          button→render();
65      }
66      loadButtons = false;
67      if (this→buttonsNPC.size() ≡ 0)
68          loadButtons = true;
69      int width = 0;
70      for (auto& action: information.actions){
71          if (action ≡ ActionsProfessionID::DepositGold ∨ action ≡ ActionsProfessi
    onID::RetireGold) {
72              if (i ≡ 0) {
73                  width = 0;
74              } else {
75                  width = 100;
76              }
77              if (loadButtons){
78                  this→buttonsNPC.push_back(createButtonAction(action,{12+width,
79                      ((this→window→getHeight()−60)/2)−HEIGTHBUTTON,WIDTHBUTTON,
    HEIGTHBUTTON}));
80              }
81              this→buttonsNPC[i]→setViewport({0,(this→window→getHeight())/2,
82                                  (this→window→getWidth()/8)*2,(this→window→getHeight()
    )/2});
83              this→buttonsNPC[i]→updatePosition({12+width,
84                              ((this→window→getHeight()−60)/2)−HEIGTHBUTTON,WIDTHBUTTON,
    HEIGTHBUTTON});
85              this→buttonsNPC[i]→render();
86              i++;
87          }
88      }
89  }
90
91
92  void BankerInterface::renderItems() {
93      bool loadButtons = false;
94      SDL_Rect src;
95      SDL_Rect dst;
96      int i = 0;
97      int width = 0;
98      std::shared_ptr<RaisedButton> button;
99      std::shared_ptr<SelectButton> selection;
100
101     this→pagMaxInBank = information.itemsInBank.size()/ITEMSBANKER+1;
102
103     if (this→buttonsItemsNPC.size() ≠ information.itemsInBank.size()) {
104         loadButtons = true;
105         this→buttonsItemsNPC.clear();
106     }
107     uint max;
108     if ((pagItemsInBank+1)*ITEMSBANKER > information.itemsInBank.size()){
109         max = information.itemsInBank.size();
110     } else {
111         max =(pagItemsInBank+1)*ITEMSBANKER;
112     }
113     int widthSegment = this→window→getWidth()/8;
114     for (uint j = pagItemsInBank*ITEMSBANKER; j < max; j++) {
115         const Texture& item = manager.getTexture(information.itemsInBank[j]);
116         src = {0,0,52,52};
117         dst = {20+(i%3)*widthSegment/2,50+(i/3)*((this→window→getHeight())/2)/
    5,widthSegment/3,widthSegment/3};
118         if (loadButtons) {
119             selection = std::make_shared<SelectButton>(&(window→getRenderer()),
    dst,manager,j);
```



```
120             this→buttonsItemsNPC.push_back(selection);
121         }
122         this→buttonsItemsNPC[i]→setViewport({0,(this→window→getHeight())/2,
123                                 (this→window→getWidth()/8)*2,(this→window→get
    Height())/2});
124         this→buttonsItemsNPC[i]→updatePosition(dst);
125         this→buttonsItemsNPC[i]→render();
126         item.render(src,dst);
127         i++;
128         if (i ≡ ITEMSBANKER)
129             break;
130     }
131     i = 0;
132     loadButtons = false;
133     if (this→buttonsNPC.size() ≡ 0)
134         loadButtons = true;
135     for (auto& action: information.actions){
136         if (action ≡ ActionsProfessionID::DepositItem ∨ action ≡ ActionsProfessi
    onID::RetireItem) {
137             if (i ≡ 0) {
138                 width = 0;
139             } else {
140                 width = 100;
141             }
142             if (loadButtons){
143                 this→buttonsNPC.push_back(createButtonAction(action,{12+width,
144                     ((this→window→getHeight()−60)/2)−HEIGTHBUTTON,WIDTHBUTTON,
    HEIGTHBUTTON}));
145             }
146             this→buttonsNPC[i]→setViewport({0,(this→window→getHeight())/2,
147                                 (this→window→getWidth()/8)*2,(this→window→getHeight()
    )/2});
148             this→buttonsNPC[i]→updatePosition({12+width,
149                             ((this→window→getHeight()−60)/2)−HEIGTHBUTTON,WIDTHBUTTON,
    HEIGTHBUTTON});
150             this→buttonsNPC[i]→render();
151             i++;
152         }
153     }
154
155     if(loadButtons) {
156         SDL_Rect rect = {(this→window→getWidth()/8)*2−40,((this→window→getHei
    ght()−60)/2)−HEIGTHBUTTON*4,20,20};
157         this→arrow = std::make_shared<ArrowButton>(&(window→getRenderer()),fon
    t,"+",rect, manager);
158     }
159     if (this→arrow ≠ nullptr){
160         this→arrow→setViewport({0,(this→window→getHeight())/2,
161                             (this→window→getWidth()/8)*2,(this→window→getHeight())/2}
    );
162         this→arrow→updatePosition({(this→window→getWidth()/8)*2−40,
163             ((this→window→getHeight()−60)/2)−HEIGTHBUTTON*4,20,20});
164         this→arrow→render();
165     }
166
167 }
168
169 void BankerInterface::render() {
170     int w,h;
171     SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
172     SDL_Rect banker = {15,15,w,h};
173     SDL_RenderCopy(&(window→getRenderer()), this→texture, NULL, &banker);
174     if (this→goldValue ≠ nullptr) {
175         SDL_DestroyTexture(this→goldValue);
176         this→goldValue = nullptr;
```

```
177     }
178     if (this→goldInBank ≠ nullptr){
179         SDL_DestroyTexture(this→goldInBank);
180         this→goldInBank = nullptr;
181     }
182     if (pageGold) {
183         renderGoldManagment();
184     } else {
185         renderItems();
186     }
187     if (this→changeScreen ≠ nullptr){
188         this→changeScreen→setViewport({0,(this→window→getHeight())/2,
189                     (this→window→getWidth()/8)*2,(this→window→getHeight())/2}
    );
190         this→changeScreen→updatePosition({(this→window→getWidth()/8)*2−40,
191             ((this→window→getHeight()−60)/2)−HEIGTHBUTTON*3,20,20});
192         this→changeScreen→render();
193     }
194 }
195
196 InputInfo BankerInterface::handleClick(int x, int y, int itemSelected) {
197     InputInfo info;
198     info.input  = InputID::nothing;
199     bool newItemSelected = false;
200     for(auto& button: buttonsItemsNPC){
201         newItemSelected = button→inside(x,y);
202         if (newItemSelected) {
203             if(itemSelectedNPC ≡ −1) {
204                 itemSelectedNPC = button→getId() ;
205             } else if (itemSelectedNPC ≡ button→getId()) {
206                 itemSelectedNPC = −1;
207             } else if (itemSelectedNPC ≠ button→getId()) {
208                 buttonsItemsNPC[itemSelectedNPC]→onClick();
209                 itemSelectedNPC = button→getId();
210             }
211         }
212     }
213     if (pageGold) {
214         if (buttonsNPC[0]→inside(x,y))
215             info = buttonsNPC[0]→onClick(amountGold);
216         if (buttonsNPC[1]→inside(x,y))
217             info = buttonsNPC[1]→onClick(amountGold);
218     } else {
219         if (buttonsNPC[0]→inside(x,y) ∧ itemSelected > 0)
220             info = buttonsNPC[0]→onClick(itemSelected);
221         if (buttonsNPC[1]→inside(x,y) ∧ itemSelectedNPC > −1) {
222             info = buttonsNPC[1]→onClick(itemSelectedNPC);
223             itemSelectedNPC = −1;
224         }
225     }
226    if (this→changeScreen ≠ nullptr ∧ this→changeScreen→inside(x,y)) {
227         this→changeScreen→onClick();
228         this→pageGold = ¬this→pageGold;
229         this→buttonsItemsNPC.clear();
230         this→buttonsNPC.clear();
231     }
232     if (this→arrows.size() ≠ 0) {
233         if (this→arrows[0]→inside(x,y)){
234             this→amountGold +=10;
235         }
236         if (this→arrows[1]→inside(x,y)) {
237             this→amountGold −=10;
238             if (this→amountGold < 0)
239                 this→amountGold = 0;
240         }
241     }
```



```
242     return info;
243 }
244
245 BankerInterface::~BankerInterface() {
246     SDL_DestroyTexture(this→inBank);
247 }
```

jul 21, 20 15:20 **ArrowButton.h** Page 1/1

```
#ifndef ARROWBUTTON_H
#define ARROWBUTTON_H

#include "Button.h"
#include "../Player.h"

class ArrowButton: public Button {
public:
    ArrowButton(SDL_Renderer* renderer, Font& font,std::string text, SDL_Rect po
sition,
     const TextureManager& manager);

    virtual bool inside(int x, int y);

    virtual void render();

    virtual void onClick();

    void setViewport(SDL_Rect viewport);

    virtual ~ArrowButton();
};

#endif
```

jul 21, 20 15:20 **ArrowButton.cpp** Page 1/1

```
#include "ArrowButton.h"

ArrowButton::ArrowButton(SDL_Renderer* renderer, Font& font,std::string text,
    SDL_Rect position, const TextureManager& manager) :
        Button(renderer,font,text,position,manager) {}

bool ArrowButton::inside(int x, int y) {
     bool inside = true;
    if (x < this→button.x+viewport.x)
        inside = false;
    if (x > this→button.x+this→button.w+viewport.x)
        inside = false;
    if (y < this→button.y+viewport.y)
        inside = false;
    if (y > this→button.y + this→button.h+viewport.y)
        inside = false;
    return inside;
}

void ArrowButton::onClick() {
    this→clicked = ¬this→clicked;
}

void ArrowButton::render() {
   int w,h;
    const Texture& background = manager.getTexture(TextureID::Button);
    SDL_QueryTexture(background.getTexture(), NULL, NULL, &w, &h);
    SDL_Rect src = {0,0,w,h};
    background.render(src,this→button);
    if(this→clicked) {
        SDL_Rect click = {button.x−1,button.y−1,button.w+2,button.h+2};
        SDL_SetRenderDrawColor(this→renderer, 0xA4, 0xA4, 0xA4, 0xAA);
      SDL_RenderDrawRect(this→renderer, &click);
        this→clicked = ¬this→clicked;
    }
     if(this→texture ≠ nullptr){
        SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
        src = {0,0,w,h};
        SDL_Rect dest = {button.x, button.y−button.h,w*2,h*2};
        SDL_RenderCopy(this→renderer,this→texture,&src,&dest);
    }
}

void ArrowButton::setViewport(SDL_Rect viewport){
    Button::setViewport(viewport);
}

ArrowButton::~ArrowButton(){}
```

jul 21, 20 15:20 **Tile.h** Page 1/1

```h
#ifndef ARGENTUM_TILE_H
#define ARGENTUM_TILE_H


#include "Texture.h"
#include "Camera.h"
#include "../common/Point.h"

class Tile {
private:
    int x;
    int y;
    int tx;
    int ty;
    int width;
    int height;
    Texture& texture;
    SDL_Rect src, dest;
public:
    Tile(int x, int y, int tx, int ty, int width, int height, Texture &texture);

    virtual ~Tile();

    void destinyAdjust(Camera& camera);

    void draw(SDL_Renderer&, Camera& camera);

};


#endif //ARGENTUM_TILE_H
```

jul 21, 20 15:20 **Tile.cpp** Page 1/1

```cpp
#include "Tile.h"

Tile::Tile(int x, int y, int tx, int ty,
        int width, int height, Texture &texture) :
        x(x), y(y), tx(tx), ty(ty),
        width(width), height(height),
        texture(texture) {}

void Tile::draw(SDL_Renderer &renderer, Camera& camera) {
    src.w = width;
    src.h = height;
    src.x = tx;
    src.y = ty;
    destinyAdjust(camera);
    texture.render(src, dest);
}

void Tile::destinyAdjust(Camera& camera) {
    this→dest.w = width*camera.getScale();
    this→dest.h = height*camera.getScale();
    this→dest.x = (x − camera.getCameraPosition().x)−(width/2)*camera.getScale(
);
    this→dest.y = (y − camera.getCameraPosition().y)−(height/2)*camera.getScale
();

}

Tile::~Tile() = default;
```

jul 21, 20 15:20 **TextureManager.h** Page 1/1

```h
#ifndef TEXTUREMANAGER_H
#define TEXTUREMANAGER_H
#include "Texture.h"
#include "TextureID.h"
#include <unordered_map>
#include <string>
#include "../common/Identificators.h"

class SDL_Renderer;
class SDL_Color;

class TextureManager {
private:
  std::unordered_map<TextureID, Texture, std::hash<TextureID>> textures;
  SDL_Renderer& renderer;

public:
  explicit TextureManager(SDL_Renderer& renderer);

  void loadTextures();

  void createTexture(TextureID id, const std::string& path, SDL_Color color);
  void createTexture(TextureID id, const std::string& path) ;

  void dropTexture(TextureID id);

  const Texture& getTexture(TextureID id) const;

  const Texture& getTexture(ItemsInventoryID id) const;
  const Texture& getTexture(BodyID id) const;
  const Texture& getTexture(HeadID id) const;
  const Texture& getTexture(WeaponID id) const;
  const Texture& getTexture(ShieldID id) const;
  const Texture& getTexture(HelmetID id) const;
  ~TextureManager();

  TextureManager(const TextureManager&) = delete;
    TextureManager &operator=(const TextureManager&) = delete;
};

#endif
```

jul 21, 20 15:20 **TextureManager.cpp** Page 1/6

```cpp
#include "TextureManager.h"
#include <utility>

TextureManager::TextureManager(SDL_Renderer& renderer) : textures(),
  renderer(renderer) {}

void TextureManager::createTexture(TextureID id, const std::string& path) {
  Texture newTexture(path, this→renderer);
  this→textures.insert(std::make_pair(id, std::move(newTexture)));
}

void TextureManager::createTexture(TextureID id, const std::string& path, SDL_Color color) {
  Texture newTexture(path, this→renderer, color);
  this→textures.insert(std::make_pair(id, std::move(newTexture)));
}

void TextureManager::dropTexture(TextureID id) {
  this→textures.erase(id);
}

const Texture& TextureManager::getTexture(TextureID id) const {
  return this→textures.at(id);
}

const Texture& TextureManager::getTexture(BodyID id) const {
  TextureID idItem = TextureID::ItemNothing;
  switch(id) {
    case BodyID::BlueCommon:
      idItem = TextureID::ItemBlueCommonBody;
      break;
    case BodyID::GreenCommon:
      idItem = TextureID::ItemGreenCommonBody;
      break;
    case BodyID::RedCommon:
      idItem = TextureID::ItemRedCommonBody;
      break;
    case BodyID::LeatherArmor:
      idItem = TextureID::ItemLeatherArmor;
      break;
    case BodyID::BlueTunic:
      idItem = TextureID::ItemBlueTunic;
      break;
    case BodyID::PlateArmor:
      idItem = TextureID::ItemPlateArmor;
      break;
    case BodyID::Ghost:
      idItem = TextureID::ItemNothing;
      break;
    default:
      idItem = TextureID::ItemNothing;
      break;
  }
  const Texture& texture = getTexture(idItem);
  return texture;
}

const Texture& TextureManager::getTexture(HeadID id) const {
  TextureID idItem = TextureID::ItemNothing;
  switch(id) {
    case HeadID::Elf:
      idItem = TextureID::ElfHead;
      break;
    case HeadID::Gnome:
      idItem = TextureID::GnomeHead;
      break;
```

```
66      case HeadID::Human:
67        idItem = TextureID::HumanHead;
68        break;
69      case HeadID::Dwarf:
70        idItem = TextureID::DwarfHead;
71        break;
72      case HeadID::Nothing:
73        idItem = TextureID::ItemNothing;
74        break;
75      default:
76        idItem = TextureID::ItemNothing;
77        break;
78    }
79    const Texture& texture = getTexture(idItem);
80    return texture;
81  }
82
83  const Texture& TextureManager::getTexture(HelmetID id) const {
84    TextureID idItem = TextureID::ItemNothing;
85    switch(id) {
86      case HelmetID::Hood:
87        idItem = TextureID::ItemHood;
88        break;
89      case HelmetID::MagicHat:
90        idItem = TextureID::ItemMagicHat;
91        break;
92      case HelmetID::IronHelmet:
93        idItem = TextureID::ItemIronHelmet;
94        break;
95      case HelmetID::Nothing:
96        idItem = TextureID::ItemNothing;
97        break;
98      default:
99        idItem = TextureID::ItemNothing;
100       break;
101   }
102   const Texture& texture = getTexture(idItem);
103   return texture;
104 }
105
106 const Texture& TextureManager::getTexture(WeaponID id) const {
107   TextureID idItem = TextureID::ItemNothing;
108   switch(id) {
109     case WeaponID::SimpleArc:
110       idItem = TextureID::ItemSimpleArc;
111       break;
112     case WeaponID::CompoundArc:
113       idItem = TextureID::ItemCompoundArc;
114       break;
115     case WeaponID::LongSword:
116       idItem = TextureID::ItemLongSword;
117       break;
118     case WeaponID::Hammer:
119       idItem = TextureID::ItemHammer;
120       break;
121     case WeaponID::Ax:
122       idItem = TextureID::ItemAx;
123       break;
124     case WeaponID::ElficFlaute:
125       idItem = TextureID::ItemElficFlaute;
126       break;
127     case WeaponID::AshStick:
128       idItem = TextureID::ItemAshStick;
129       break;
130     case WeaponID::Crosier:
131       idItem = TextureID::ItemCrosier;
```



```
132       break;
133     case WeaponID::GnarledStick:
134       idItem = TextureID::ItemGnarledStick;
135       break;
136     case WeaponID::Nothing:
137       idItem = TextureID::ItemNothing;
138       break;
139     default:
140       idItem = TextureID::ItemNothing;
141       break;
142   }
143   const Texture& texture = getTexture(idItem);
144   return texture;
145 }
146
147 const Texture& TextureManager::getTexture(ShieldID id) const {
148   TextureID idItem = TextureID::ItemNothing;
149   switch(id) {
150     case ShieldID::IronShield:
151       idItem = TextureID::ItemIronShield;
152       break;
153     case ShieldID::TurtleShield:
154       idItem = TextureID::ItemTurtleShield;
155       break;
156     case ShieldID::Nothing:
157       idItem = TextureID::ItemNothing;
158       break;
159   }
160   const Texture& texture = getTexture(idItem);
161   return texture;
162 }
163
164 const Texture& TextureManager::getTexture(ItemsInventoryID id) const {
165   TextureID idText = TextureID::ItemNothing;
166   switch(id) {
167     case ItemsInventoryID::SimpleArc:
168       idText = TextureID::ItemSimpleArc;
169       break;
170     case ItemsInventoryID::CompoundArc:
171       idText = TextureID::ItemCompoundArc;
172       break;
173     case ItemsInventoryID::LongSword:
174       idText = TextureID::ItemLongSword;
175       break;
176     case ItemsInventoryID::Hammer:
177       idText = TextureID::ItemHammer;
178       break;
179     case ItemsInventoryID::Ax:
180       idText = TextureID::ItemAx;
181       break;
182     case ItemsInventoryID::ElficFlaute:
183       idText = TextureID::ItemElficFlaute;
184       break;
185     case ItemsInventoryID::AshStick:
186       idText = TextureID::ItemAshStick;
187       break;
188     case ItemsInventoryID::Crosier:
189       idText = TextureID::ItemCrosier;
190       break;
191     case ItemsInventoryID::GnarledStick:
192       idText = TextureID::ItemGnarledStick;
193       break;
194     case ItemsInventoryID::IronShield:
195       idText = TextureID::ItemIronShield;
196       break;
197     case ItemsInventoryID::TurtleShield:
```

```
198         idText = TextureID::ItemTurtleShield;
199         break;
200       case ItemsInventoryID::Hood:
201         idText = TextureID::ItemHood;
202         break;
203       case ItemsInventoryID::MagicHat:
204         idText = TextureID::ItemMagicHat;
205         break;
206       case ItemsInventoryID::IronHelmet:
207         idText = TextureID::ItemIronHelmet;
208         break;
209       case ItemsInventoryID::BlueCommon:
210         idText = TextureID::ItemBlueCommonBody;
211         break;
212       case ItemsInventoryID::GreenCommon:
213         idText = TextureID::ItemGreenCommonBody;
214         break;
215       case ItemsInventoryID::RedCommon:
216         idText = TextureID::ItemRedCommonBody;
217         break;
218       case ItemsInventoryID::LeatherArmor:
219         idText = TextureID::ItemLeatherArmor;
220         break;
221       case ItemsInventoryID::BlueTunic:
222         idText = TextureID::ItemBlueTunic;
223         break;
224       case ItemsInventoryID::PlateArmor:
225         idText = TextureID::ItemPlateArmor;
226         break;
227       case ItemsInventoryID::HealthPotion:
228         idText = TextureID::HealthPotion;
229         break;
230       case ItemsInventoryID::ManaPotion:
231         idText = TextureID::ManaPotion;
232         break;
233           case ItemsInventoryID::Gold:
234               idText = TextureID::Gold;
235               break;
236       case ItemsInventoryID::Nothing:
237         idText = TextureID::ItemNothing;
238         break;
239       default:
240         idText = TextureID::ItemNothing;
241         break;
242     }
243     const Texture& texture = getTexture(idText);
244     return texture;
245 }
246
247 void TextureManager::loadTextures() {
248     SDL_Color textColor = {0x0, 0x0, 0x0};
249     std::string path(ROOT_DIR);
250     createTexture(TextureID::PresentationImage, path + "/assets/img/ImagenPresentacion.jpg")
    ;
251     createTexture(TextureID::LobbyBackground, path + "/assets/img/Fondo Inicio.jpg");
252     createTexture(TextureID::ZombieHead, path + "/assets/img/Zombie Cabeza.png",textColor
    );
253     createTexture(TextureID::ElfHead,path + "/assets/img/Cabeza Elfo.png", textColor);
254     createTexture(TextureID::HumanHead, path + "/assets/img/Cabeza Humano.png",textColor)
    ;
255     createTexture(TextureID::DwarfHead, path + "/assets/img/Cabeza Enano.png",textColor);
256     createTexture(TextureID::GnomeHead, path + "/assets/img/Cabeza Gnomo.png",textColor)
    ;
257     createTexture(TextureID::PriestHead,path + "/assets/img/Sacerdote Cabeza Sprite.png", tex
    tColor);
258     createTexture(TextureID::ZombieBody, path + "/assets/img/Zombie Cuerpo Sprite.png",textC
```



```
    olor);
259     createTexture(TextureID::BlueTunic,path + "/assets/img/Tunica Azul Sprite.png", textColo
    r);
260     createTexture(TextureID::PriestBody, path + "/assets/img/Sacerdote Sprite.png",textColor
    );
261     createTexture(TextureID::LeatherArmor, path + "/assets/img/Armadura de Cuero Sprite.png",
    textColor);
262     createTexture(TextureID::PlateArmor, path + "/assets/img/Armadura de Placas Sprite.png",te
    xtColor);
263     createTexture(TextureID::BlueCommonBody, path + "/assets/img/Vestimenta Comun azul Sprite.
    png",textColor);
264     createTexture(TextureID::GreenCommonBody, path + "/assets/img/Vestimenta Comun verde Spr
    ite.png",textColor);
265     createTexture(TextureID::RedCommonBody, path + "/assets/img/Vestimenta Comun roja Sprite.p
    ng",textColor);
266     createTexture(TextureID::TurtleShield, path + "/assets/img/Escudo de Tortuga Sprite.png",t
    extColor);
267     createTexture(TextureID::IronShield, path + "/assets/img/Escudo de Hierro Sprite.png",text
    Color);
268     createTexture(TextureID::IronHelmet, path + "/assets/img/Casco de Hierro Sprite.png", tex
    tColor);
269     createTexture(TextureID::Hood, path + "/assets/img/Capucha Sprite.png",textColor);
270     createTexture(TextureID::MagicHat, path + "/assets/img/Sombrero Magico Sprite.png", text
    Color);
271     createTexture(TextureID::AshStick, path + "/assets/img/Vara de Fresno Sprite.png", textCo
    lor);
272     createTexture(TextureID::LongSword, path + "/assets/img/Espada Larga Sprite.png", textCo
    lor);
273     createTexture(TextureID::Hammer, path + "/assets/img/Martillo Sprite.png", textColor);
274     createTexture(TextureID::SimpleArc, path + "/assets/img/Arco Simple Sprite.png", textCol
    or);
275     createTexture(TextureID::CompoundArc, path + "/assets/img/Arco Compuesto Sprite.png", te
    xtColor);
276     createTexture(TextureID::GnarledStick, path + "/assets/img/Baston Nudoso Sprite.png", te
    xtColor);
277     createTexture(TextureID::Crosier, path + "/assets/img/Baculo Engarzado Sprite.png", textC
    olor);
278     createTexture(TextureID::Ax, path + "/assets/img/Hacha Sprite.png", textColor);
279     createTexture(TextureID::ElficFlaute, path + "/assets/img/Flauta Elfica Sprite.png", textC
    olor);
280     createTexture(TextureID::Spider, path + "/assets/img/AraÃ±a Sprite.png",textColor);
281     createTexture(TextureID::Skeleton, path + "/assets/img/Esqueleto Sprite.png", textColor)
    ;
282     createTexture(TextureID::Banker, path + "/assets/img/Banquero Sprite.png", textColor);
283     createTexture(TextureID::Merchant, path + "/assets/img/Comerciante Sprite.png", textColo
    r);
284     createTexture(TextureID::Goblin, path + "/assets/img/Goblin.png", textColor);
285     createTexture(TextureID::Ghost, path + "/assets/img/Fantasma Sprite.png", textColor);
286     createTexture(TextureID::ManaPotion, path + "/assets/img/Pocion Mana.png", textColor)
    ;
287     createTexture(TextureID::HealthPotion, path + "/assets/img/Pocion Vida.png", textColo
    r);
288     createTexture(TextureID::Gold, path + "/assets/img/Oro.png", textColor);
289     createTexture(TextureID::ItemHood, path + "/assets/img/Capucha.png", textColor);
290     createTexture(TextureID::ItemIronHelmet, path + "/assets/img/Casco de Hierro.png", text
    Color);
291     createTexture(TextureID::ItemMagicHat, path + "/assets/img/Sombrero Magico.png", textC
    olor);
292     createTexture(TextureID::ItemIronShield, path + "/assets/img/Escudo de Hierro.png", tex
    tColor);
293     createTexture(TextureID::ItemTurtleShield, path + "/assets/img/Escudo de Tortuga.png",
    textColor);
294     createTexture(TextureID::ItemAx, path + "/assets/img/Hacha.png", textColor);
295     createTexture(TextureID::ItemSimpleArc, path + "/assets/img/Arco Simple.png", textCol
    or);
296     createTexture(TextureID::ItemCompoundArc, path + "/assets/img/Arco Compuesto.png", te
```

jul 21, 20 15:20 **TextureManager.cpp** Page 6/6

```
    xtColor);
297   createTexture(TextureID::ItemHammer, path + "/assets/img/Martillo.png", textColor);
298   createTexture(TextureID::ItemLongSword, path + "/assets/img/Espada Larga.png", textCol
    or);
299   createTexture(TextureID::ItemAshStick, path + "/assets/img/Vara de Fresno.png", textCol
    or);
300   createTexture(TextureID::ItemCrosier, path + "/assets/img/Baculo Engarzado.png", textC
    olor);
301   createTexture(TextureID::ItemGnarledStick, path + "/assets/img/Baston Nudoso.png", te
    xtColor);
302   createTexture(TextureID::ItemElficFlaute, path + "/assets/img/Flauta Elfica.png", textC
    olor);
303   createTexture(TextureID::ItemGreenCommonBody, path + "/assets/img/Vestimenta Comun ver
    de.png", textColor);
304   createTexture(TextureID::ItemBlueCommonBody, path + "/assets/img/Vestimenta Comun azul.
    png", textColor);
305   createTexture(TextureID::ItemRedCommonBody, path + "/assets/img/Vestimenta Comun roja.pn
    g", textColor);
306   createTexture(TextureID::ItemBlueTunic, path + "/assets/img/Tunica Azul.png", textCol
    or);
307   createTexture(TextureID::ItemLeatherArmor, path + "/assets/img/Armadura de Cuero.png",
     textColor);
308   createTexture(TextureID::ItemPlateArmor, path + "/assets/img/Armadura de Placas.png", t
    extColor);
309   createTexture(TextureID::TopBar, path + "/assets/img/Fondo Barra Superior.jpg");
310   createTexture(TextureID::Star, path + "/assets/img/Estrella.jpg", textColor);
311   createTexture(TextureID::StatsBackground, path + "/assets/img/StatsBackground.jpg");
312   createTexture(TextureID::ItemNothing, path + "/assets/img/RecuadroItems.jpg");
313   createTexture(TextureID::MeditateEffect, path + "/assets/img/Efecto Meditar.png", textC
    olor);
314   createTexture(TextureID::HitEffect, path + "/assets/img/Efecto Golpe.png", textColor);
315   createTexture(TextureID::MagicArrowEffect, path + "/assets/img/Efecto Flecha Magica Sprite
    .png",textColor);
316   createTexture(TextureID::MissileEffect, path + "/assets/img/Efecto Misil Sprite.png",text
    Color);
317   createTexture(TextureID::ExplotionEffect, path + "/assets/img/Efecto Explosion Sprite.png"
    ,textColor);
318   createTexture(TextureID::HealthEffect, path + "/assets/img/Efecto Curar Sprite.png",textC
    olor);
319   createTexture(TextureID::Button, path + "/assets/img/Boton.bmp");
320 }
321
322 TextureManager::~TextureManager() = default;
```

jul 21, 20 15:20 **TextureID.h** Page 1/2

```
1  #ifndef TEXTUREID_H
2  #define TEXTUREID_H
3
4  enum class TextureID {
5    LobbyBackground,
6    PresentationImage,
7    TopBar,
8    StatsBackground,
9    Button,
10   Star,
11   ZombieHead,
12   ElfHead,
13   HumanHead,
14   DwarfHead,
15   GnomeHead,
16   PriestHead,
17   ZombieBody,
18   PriestBody,
19   BlueTunic,
20   LeatherArmor,
21   PlateArmor,
22   BlueCommonBody,
23   GreenCommonBody,
24   RedCommonBody,
25   TurtleShield,
26   IronShield,
27   IronHelmet,
28   MagicHat,
29   Hood,
30   AshStick,
31   SimpleArc,
32   CompoundArc,
33   LongSword,
34   Hammer,
35   Crosier,
36   GnarledStick,
37   Ax,
38   ElficFlaute,
39   Spider,
40   Skeleton,
41   Goblin,
42   Ghost,
43   Merchant,
44   Banker,
45   HealthEffect,
46   ExplotionEffect,
47   MagicArrowEffect,
48   MissileEffect,
49   MeditateEffect,
50   HitEffect,
51   ManaPotion,
52   HealthPotion,
53   Gold,
54   ItemSimpleArc,
55   ItemCompoundArc,
56   ItemLongSword,
57   ItemHammer,
58   ItemCrosier,
59   ItemGnarledStick,
60   ItemAx,
61   ItemAshStick,
62   ItemElficFlaute,
63   ItemTurtleShield,
64   ItemIronShield,
65   ItemIronHelmet,
66   ItemMagicHat,
```

jul 21, 20 15:20 **TextureID.h** Page 2/2

```
    ItemHood,
    ItemBlueTunic,
    ItemLeatherArmor,
    ItemPlateArmor,
    ItemBlueCommonBody,
    ItemGreenCommonBody,
    ItemRedCommonBody,
    ItemNothing,
};

#endif
```

jul 21, 20 15:20 **Texture.h** Page 1/1

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

class Texture {
private:
  SDL_Texture* texture = nullptr;
  int width;
  int height;
  SDL_Renderer* renderer = nullptr;
  SDL_Texture* loadTexture(const std::string &fileName, SDL_Color colorKey);

public:
  Texture(const std::string &fileName, SDL_Renderer& renderer, SDL_Color colorKe
y);
  Texture(const std::string &fileName, SDL_Renderer& renderer);
  Texture(Texture& other);
  ~Texture();

  int render(const SDL_Rect& source, const SDL_Rect& destiny) const;

  int render() const;

    int getWidth() const;

  SDL_Texture* getTexture() const;
};

#endif
```

jul 21, 20 15:20 **Texture.cpp** Page 1/2

```cpp
#include "Texture.h"
#include "Window.h"
#include <SDL2/SDL_image.h>
#include "../common/Exception.h"
#include <iostream>

Texture::Texture(const std::string &fileName, SDL_Renderer& renderer, SDL_Color colorKey) {
  this→renderer = &renderer;
  this→texture = loadTexture(fileName, colorKey);
}

Texture::Texture(const std::string &fileName, SDL_Renderer& renderer) {
  this→renderer = &renderer;
  if (¬this→texture) {
    this→texture = IMG_LoadTexture(this→renderer, fileName.c_str());
  }
  if (¬this→texture){
    throw Exception("Fail IMG_LoadTexture: %s ", IMG_GetError());
  }
  int h,w;
  SDL_QueryTexture(this→texture, NULL, NULL, &w, &h);
  this→width = w;
  this→height = h;
}


Texture::Texture(Texture ∧other) {
    std::swap(this→texture, other.texture);
    std::swap(this→renderer, other.renderer);
    std::swap(this→height, other.height);
    std::swap(this→width, other.width);
}

SDL_Texture* Texture::loadTexture(const std::string &fileName, SDL_Color colorKey) {
  SDL_Surface* surface = IMG_Load(fileName.c_str());
  if (¬surface) {
    throw Exception("Fail IMG_Load: %s", IMG_GetError());
  }
  SDL_SetColorKey( surface, SDL_TRUE,
    SDL_MapRGB(surface→format, colorKey.r, colorKey.g, colorKey.b));
  this→width = surface→w;
  this→height = surface→h;
  this→texture = SDL_CreateTextureFromSurface(this→renderer,surface);

  if (¬this→texture){
    throw Exception("Fail SDL_CreateTextureFromSurface: %s", SDL_GetError());
  }
  SDL_FreeSurface(surface);
  return this→texture;
}

int Texture::render() const {
  return SDL_RenderCopy(this→renderer, this→texture, NULL, NULL);
}

int Texture::render(const SDL_Rect& source, const SDL_Rect& destiny) const {
  return SDL_RenderCopy(this→renderer, this→texture, &source, &destiny);
}

Texture::~Texture() {
  if (this→texture) {
    SDL_DestroyTexture(this→texture);
    this→texture = nullptr;
  }
```

jul 21, 20 15:20 **Texture.cpp** Page 2/2

```cpp
}

int Texture::getWidth() const {
    return width;
}

SDL_Texture* Texture::getTexture() const {
  return this→texture;
}
```

jul 21, 20 15:20 **Receiver.h** Page 1/1

```h
#ifndef RECEIVER_H
#define RECEIVER_H

#include "../common/Thread.h"
#include "../common/DataQueue.h"
#include "../common/CommunicationProtocol.h"

class Receiver: public Thread {
private:
    DataQueue& queue;
    std::atomic<bool> keepTalking;
    CommunicationProtocol& protocol;
public:

    Receiver(CommunicationProtocol& protocol, DataQueue& queue);

    virtual void run();

    bool is_alive() const;

    void stop();

    virtual ~Receiver();

};

#endif
```

jul 21, 20 15:20 **Receiver.cpp** Page 1/1

```cpp
#include "Receiver.h"
#include <vector>
#include <iostream>
#include "../common/Message.h"
#include "../common/SocketException.h"

#define UNKNOW_ERROR "Unknow Error"
#define ERRORSOCKET "Error en la comunicaciÃ³n en Receiver::run() "
#define ERRORRECEIVER "Error en Receiver::run() "

Receiver::Receiver(CommunicationProtocol& protocol, DataQueue& queue) :
    queue(queue), keepTalking(true), protocol(protocol) {}

void Receiver::run() {
    Message msg;
    while (this→keepTalking){
        try {
            msg = this→protocol.receive();
            this→queue.push(msg);
        }  catch (const SocketException& e) {
            std::cerr << ERRORSOCKET << e.what() << std::endl;
            this→keepTalking = false;
        } catch(const std::exception& e){
            std::cerr << ERRORRECEIVER << e.what() << std::endl;
            this→keepTalking = false;
        }catch (...) {
            this→keepTalking = false;
            std::cerr << UNKNOW_ERROR << std::endl;
        }
    }
}

bool Receiver::is_alive() const {
    return this→keepTalking;
}

void Receiver::stop() {
    this→keepTalking = false;
    this→protocol.stop();
}

Receiver::~Receiver() {}
```

jul 21, 20 15:20 **Presentation.h** Page 1/1

```
#ifndef PRESENTATION_H
#define PRESENTATION_H

#include "Window.h"
#include "TextureManager.h"

class Presentation {
private:
    Window& window;
    TextureManager& manager;
public:
    Presentation(Window& window, TextureManager& manager);

    bool run();

    ~Presentation();

};

#endif
```

jul 21, 20 15:20 **Presentation.cpp** Page 1/1

```
#include "Presentation.h"
#include <SDL2/SDL.h>
#include "Texture.h"
#include "Font.h"

Presentation::Presentation(Window& window, TextureManager& manager) :
    window(window), manager(manager) {}

bool Presentation::run() {
    bool quit = false;
    bool closeProgram = false;
  int width_text, height_text;
    SDL_Event event;
    SDL_Rect dest;
    const Texture& fondo = this→manager.getTexture(TextureID::PresentationImage
);
    std::string fuente = "assets/font/Prince Valiant.ttf";

    SDL_Color textColor = {0xFF, 0xFF, 0xFF};
    Font font(fuente, 18, textColor);
  font.setColor(textColor);
  SDL_Texture* messageTexture = font.createText("Presionar ENTER para continuar",
                                &(window.getRenderer()), &width_text, &height_te
xt);

    while (¬quit) {

    while (SDL_PollEvent(&event) ≠ 0) {
      if (event.type ≡ SDL_QUIT) {
        quit = true;
                closeProgram = true;
      } else if(event.type ≡ SDL_KEYDOWN) {
        if (event.key.keysym.sym  ≡ SDLK_RETURN){
          quit = true;
      }
        }
        window.clearScreen();
        fondo.render();
        dest = {(window.getWidth() − width_text) / 2, (window.getHeight() − heig
ht_text)* 6/7, width_text, height_text};
    SDL_RenderCopy(&(window.getRenderer()), messageTexture, NULL, &dest);
    window.render();
        }
    }
    font.deleteText(messageTexture);
    return closeProgram;
}

Presentation::~Presentation(){}
```

jul 21, 20 15:20 **Player.h** Page 1/1

```
#ifndef PLAYER_H
#define PLAYER_H

#include "Character.h"
#include "Camera.h"
#include "TextureManager.h"
#include "MusicManager.h"
#include "../common/Point.h"
#include "characterStates/CharacterState.h"
#include "../common/Identificators.h"
#include "../common/PlayerInfo.h"
#include <memory>
#include <string>

union SDL_Event;


class Player : public Character {
private:
  Point center;
  PlayerInfo playerInfo;
  std::shared_ptr<CharacterState> state = nullptr;
  bool playLowLife{false};

  void setState(CharacterStateID newState);
  void playEffectLowLife();

public:
  Player(const TextureManager& manager, const PlayerInfo& playerInfo, const MusicManager& mixer);

  virtual void render(Camera& camera);
  virtual void update(double dt);

  void updatePlayerInfo(const PlayerInfo& info);

  InputInfo handleEvent(SDL_Event& event, Camera& camera);
  InputInfo dropItem(int itemNumber);
  InputInfo selectItem(int itemNumber);
  InputInfo resurrect();
  InputInfo cure();
  InputInfo buy(int itemNumber);
  InputInfo sell(int itemNumber);
  InputInfo deposit(int information, bool isItem);
  InputInfo retire(int information, bool isItem);
  InputInfo unequipItem(int itemNumber);
  Point* getCenter();

  uint getLevel() const;
  uint getHealth() const;
  uint getMana() const;
  uint getGold() const;
  uint getMaxMana() const;
  uint getMaxHealth() const;
  uint getSafeGold() const;
  uint getExp() const;
  uint getMaxExp() const;
  std::string getInventory() const;
  CharacterStateID& getState() const;
  PlayerInfo getInfo() const;

  ~Player();
};

#endif
```

jul 21, 20 15:20 **Player.cpp** Page 1/5

```
#include "Player.h"
#include <SDL2/SDL.h>
#include <iostream>
#include "characterStates/StillState.h"
#include "characterStates/MoveState.h"
#include "characterStates/MeditateState.h"
#include "characterStates/InteractState.h"
#include "characterStates/AttackState.h"
#include "characterStates/ResurrectState.h"
#include "Items/Animation.h"
#include "Items/MeditateAnimation.h"


Player::Player(const TextureManager& manager, const PlayerInfo& playerInfo,
  const MusicManager& mixer) : Character(playerInfo.getX(),playerInfo.getY(),
  playerInfo.getId(),manager,mixer), center(playerInfo.getX(),playerInfo.getY()),
   playerInfo(playerInfo){

  this→direction = playerInfo.getDirection();
  this→frameHead = 0;
  this→state = std::shared_ptr<CharacterState>(new StillState());
  setArmor(playerInfo.getBodyID());
  setHead(playerInfo.getHeadID());
  setHelmet(playerInfo.getHelmetID());
  setShield(playerInfo.getShieldID());
  setWeapon(playerInfo.getWeaponID());
}

void Player::render(Camera& camera) {
  Character::render(camera);
  playEffectLowLife();
}


void Player::update(double dt) {
  if(this→state ≠ nullptr ∧
     (this→state→getState() ≡ CharacterStateID::Move ∨ this→state→getState() ≡ CharacterStateID::Resurrect)) {
    Point aux(posX, posY);
    this→center = aux;
    this→body→update(dt,direction);
    if (this→weapon ≠ nullptr)
      this→weapon→update(dt,direction);
    if (this→shield ≠ nullptr)
      this→shield→update(dt,direction);
  }
  if (this→animation ≠ nullptr)
    this→animation→update();
}

void Player::updatePlayerInfo(const PlayerInfo& info) {
  this→posX = info.getX();
  this→posY = info.getY();
  this→direction = info.getDirection();
  this→playerInfo = info;
  setState(info.getState());
  setArmor(info.getBodyID());
  setHead(info.getHeadID());
  setHelmet(info.getHelmetID());
  setShield(info.getShieldID());
  setWeapon(info.getWeaponID());
  setFrameHead();
  setAnimation(info.getAttackWeapon());
}

```

jul 21, 20 15:20 **Player.cpp** Page 2/5

```
65  void Player::playEffectLowLife() {
66    if(this→playerInfo.getLife() < this→playerInfo.getMaxLife()*0.1 ∧
67      this→playerInfo.getLife() ≠ 0 ∧ playLowLife) {
68      const Effect& effect = mixer.getEffect(MusicID::Heart);
69      effect.playEffect(−1);
70      playLowLife = false;
71    } else if (this→playerInfo.getLife() ≥ this→playerInfo.getMaxLife()*0.1 ∧
72     ¬playLowLife ∧ this→playerInfo.getLife() ≡ 0) {
73      const Effect& effect = mixer.getEffect(MusicID::Heart);
74      effect.pause();
75      playLowLife = true;
76    }
77  }
78
79  InputInfo Player::dropItem(int itemNumber) {
80    InputInfo info = this→state→dropItem(*this, itemNumber);
81    return info;
82  }
83
84  InputInfo Player::selectItem(int itemNumber) {
85    InputInfo info = this→state→selectItem(*this, itemNumber);
86    return info;
87  }
88
89  InputInfo Player::resurrect() {
90    InputInfo info = this→state→resurrect(*this);
91    return info;
92  }
93
94  InputInfo Player::cure() {
95    InputInfo info = this→state→cure(*this);
96    return info;
97  }
98
99  InputInfo Player::buy(int itemNumber) {
100   InputInfo info = this→state→buyItem(*this,itemNumber);
101   return info;
102 }
103
104 InputInfo Player::deposit(int information, bool isItem) {
105   InputInfo info = this→state→deposit(*this,information,isItem);
106   return info;
107 }
108
109 InputInfo Player::retire(int information, bool isItem) {
110   InputInfo info = this→state→retire(*this,information,isItem);
111   return info;
112 }
113
114 InputInfo Player::sell(int itemNumber) {
115   InputInfo info = this→state→sellItem(*this,itemNumber);
116   return info;
117 }
118
119 InputInfo Player::unequipItem(int itemNumber) {
120   InputInfo info = this→state→unequipItem(*this,itemNumber);
121   return info;
122 }
123
124 InputInfo Player::handleEvent(SDL_Event& event, Camera& camera) {
125   InputInfo input;
126   if(event.type ≡ SDL_KEYDOWN) {
127       switch (event.key.keysym.sym) {
128             case SDLK_w:
129                 input = this→state→moveUp(*this);
130                 break;
```

jul 21, 20 15:20 **Player.cpp** Page 3/5

```
131             case SDLK_s:
132                 input = this→state→moveDown(*this);
133                 break;
134             case SDLK_a:
135                 input = this→state→moveLeft(*this);
136                 break;
137             case SDLK_d:
138                 input = this→state→moveRight(*this);
139                 break;
140             case SDLK_r:
141                 input = this→state→resurrect(*this);
142                 break;
143             case SDLK_q:
144                 input = this→state→takeItem(*this);
145                 break;
146             case SDLK_h:
147                 input = this→state→cure(*this);
148                 break;
149             case SDLK_e:
150                 input = this→state→meditate(*this);
151                 break;
152             case SDLK_1:
153                 input = this→state→selectItem(*this, 1);
154                 break;
155             case SDLK_2:
156                 input = this→state→selectItem(*this, 2);
157                 break;
158             case SDLK_3:
159                 input = this→state→selectItem(*this, 3);
160                 break;
161             case SDLK_4:
162                 input = this→state→selectItem(*this, 4);
163                 break;
164             case SDLK_5:
165                 input = this→state→selectItem(*this, 5);
166                 break;
167             case SDLK_6:
168                 input = this→state→selectItem(*this, 6);
169                 break;
170             case SDLK_7:
171                 input = this→state→selectItem(*this, 7);
172                 break;
173             case SDLK_8:
174                 input = this→state→selectItem(*this, 8);
175                 break;
176             case SDLK_9:
177                 input = this→state→selectItem(*this, 9);
178                 break;
179         }
180     } else if (event.type ≡ SDL_KEYUP) {
181     switch(event.key.keysym.sym) {
182         case SDLK_w:
183         input = this→state→stopMove(*this);
184         break;
185             case SDLK_s:
186                 input = this→state→stopMove(*this);
187                 break;
188             case SDLK_a:
189                 input = this→state→stopMove(*this);
190                 break;
191             case SDLK_d:
192                  input = this→state→stopMove(*this);
193                  break;
194     }
195   } else if (event.type ≡ SDL_MOUSEBUTTONDOWN) {
196         int x,y;
```

```
197             SDL_GetMouseState(&x, &y);
198             if (camera.clickInMap(Point(x,y))){
199               Point coord = camera.calculateGlobalPosition(Point(x,y));
200               input = this→state→selectTarget(*this, coord);
201             }
202     }
203       return input;
204   }
205
206   void Player::setState(CharacterStateID newState) {
207     if(this→state ≠ nullptr ∧
208        this→state→getState() ≡ CharacterStateID::Meditate ∧
209        newState ≠ CharacterStateID::Meditate)
210        this→animation = nullptr;
211     if(this→state ≡ nullptr ∨ this→state→getState() ≠ newState) {
212       switch (newState) {
213         case CharacterStateID::Still:
214           this→state = std::shared_ptr<CharacterState>(new StillState());
215           break;
216         case CharacterStateID::Interact:
217           this→state = std::shared_ptr<CharacterState>(new InteractState());
218           break;
219         case CharacterStateID::Move:
220           this→state = std::shared_ptr<CharacterState>(new MoveState());
221           break;
222         case CharacterStateID::Meditate:
223           this→state = std::shared_ptr<CharacterState>(new MeditateState());
224           this→animation = std::shared_ptr<Animation>(new MeditateAnimation(this
    →manager,mixer.getEffect(MusicID::Meditation)));
225           break;
226         case CharacterStateID::Attack:
227           this→state = std::shared_ptr<CharacterState>(new AttackState());
228           break;
229         case CharacterStateID::Resurrect:
230           this→state = std::shared_ptr<CharacterState>(new ResurrectState());
231           break;
232       }
233     }
234   }
235
236   Point* Player::getCenter() {
237     return &center;
238   }
239
240   uint Player::getLevel() const {
241     return this→playerInfo.getLevel();
242   }
243
244   uint Player::getHealth() const {
245     return this→playerInfo.getLife();
246   }
247
248   uint Player::getMana() const{
249     return this→playerInfo.getMana();
250   }
251
252   uint Player::getGold() const {
253     return this→playerInfo.getGoldAmount();
254   }
255
256   uint Player::getMaxHealth() const {
257     return this→playerInfo.getMaxLife();
258   }
259
260   uint Player::getMaxMana()const {
261     return this→playerInfo.getMaxMana();
```



```
262   }
263
264   uint Player::getSafeGold() const {
265     return this→playerInfo.getSafeGold();
266   }
267
268   uint Player::getExp() const {
269     return this→playerInfo.getExp();
270   }
271
272   uint Player::getMaxExp() const {
273     return this→playerInfo.getMaxExp();
274   }
275
276   std::string Player::getInventory() const {
277     return this→playerInfo.getInventory();
278   }
279
280   CharacterStateID& Player::getState() const {
281     return this→state→getState();
282   }
283
284   PlayerInfo Player::getInfo() const {
285     return this→playerInfo;
286   }
287
288   Player::~Player()= default;
```

jul 21, 20 15:20 **NPC.h** Page 1/1

```
#ifndef NPC_H
#define NPC_H

#include "Character.h"
#include "Items/Item.h"
#include "TextureManager.h"
#include "MusicManager.h"
#include "../common/Identificators.h"
#include "../common/GameObjectInfo.h"
#include "characterStates/CharacterState.h"
#include <memory>

class NPC : public Character {
private:
    std::shared_ptr<CharacterState> state = nullptr;

    bool aItem{false};
    std::shared_ptr<Item> item = nullptr;
    void setState(CharacterStateID newState);
    void setItem(ItemsInventoryID itemId);
    MusicID selectSound();

public:
    NPC(const TextureManager& manager, const GameObjectInfo& gameObjectInfo, const MusicManager& mixer);

    virtual void render(Camera& camera);
  virtual void update(double dt);
    bool isItem() const;
    void updatePlayerInfo(const GameObjectInfo& info);
    CharacterStateID& getState() const;
    ~NPC();

};

#endif
```

jul 21, 20 15:20 **NPC.cpp** Page 1/3

```
#include "NPC.h"

#include <memory>
#include "characterStates/StillState.h"
#include "characterStates/MoveState.h"
#include "characterStates/MeditateState.h"
#include "characterStates/InteractState.h"
#include "characterStates/AttackState.h"
#include "characterStates/ResurrectState.h"
#include "../common/Random.h"
#include "Items/Animation.h"
#include "Items/MeditateAnimation.h"

NPC::NPC(const TextureManager& manager, const GameObjectInfo& gameObjectInfo,
  const MusicManager& mixer): Character(gameObjectInfo.getX(), gameObjectInfo.getY(),
  gameObjectInfo.getId(),manager,mixer){
    this→direction = gameObjectInfo.getDirection();
    setState(gameObjectInfo.getState());
    setFrameHead();
    setArmor(gameObjectInfo.getBodyID());
    setHead(gameObjectInfo.getHeadID());
    setHelmet(gameObjectInfo.getHelmetID());
    setShield(gameObjectInfo.getShieldID());
    setWeapon(gameObjectInfo.getWeaponID());
}

void NPC::render(Camera& camera) {
  if (¬aItem) {
    int distance = camera.distanceFromTarget(this→getPosition());
    Character::render(camera);
    if(distance < 400){
      MusicID effectId = selectSound();
      if (Random::get(0,600) ≡ 1 ∧ effectId ≠ MusicID::Nothing) {
        const Effect& effect = mixer.getEffect(effectId);
        if (distance > 255)
          distance = 255;
        effect.setDistance(distance);
        effect.playEffect(0,64);
      }
    }
  } else {
    if (this→item ≠ nullptr)
      this→item→render(int(posX−camera.getCameraPosition().x), int(posY−camera.getCameraPosition().y));
  }
}

void NPC::update(double dt) {
  if(this→state ≠ nullptr ∧ this→state→getState() ≡ CharacterStateID::Move) {
    if(this→body ≠nullptr)
      this→body→update(dt,direction);
    if (this→weapon ≠ nullptr)
        this→weapon→update(dt,direction);
    if (this→shield ≠ nullptr)
        this→shield→update(dt,direction);
  }
  if (this→animation ≠ nullptr)
    this→animation→update();
}

void NPC::updatePlayerInfo(const GameObjectInfo &info) {
    this→posX = info.getX();
    this→posY = info.getY();
    this→direction = info.getDirection();
    setState(info.getState());
```

```
65      setArmor(info.getBodyID());
66      setHead(info.getHeadID());
67      setHelmet(info.getHelmetID());
68      setShield(info.getShieldID());
69      setWeapon(info.getWeaponID());
70      setFrameHead();
71      setAnimation(info.getAttackWeapon());
72      if (info.isItem())
73        setItem(info.getItemID());
74  }
75
76  bool NPC::isItem() const {
77    return this→aItem;
78  }
79
80  void NPC::setItem(ItemsInventoryID itemId) {
81    this→aItem = true;
82    this→item = std::make_shared<Item>(manager.getTexture(itemId),32,32);
83  }
84
85  MusicID NPC::selectSound() {
86    MusicID effectId = MusicID::Nothing;
87    if (body ≠ nullptr) {
88        switch (this→body→getId()) {
89            case BodyID::Goblin:
90                effectId = MusicID::Goblin;
91                break;
92            case BodyID::Skeleton:
93                effectId = MusicID::Skeleton;
94                break;
95            case BodyID::Zombie:
96                effectId = MusicID::Zombie;
97                break;
98            default:
99                effectId = MusicID::Nothing;
100       }
101   }
102   return effectId;
103 }
104
105 void NPC::setState(CharacterStateID newState) {
106   if(this→state ≠ nullptr ∧
107      this→state→getState() ≡ CharacterStateID::Meditate ∧
108      newState ≠ CharacterStateID::Meditate)
109      this→animation = nullptr;
110   if(this→state ≡ nullptr ∨ this→state→getState() ≠ newState) {
111     switch (newState) {
112       case CharacterStateID::Still:
113         this→state = std::shared_ptr<CharacterState>(new StillState());
114         break;
115       case CharacterStateID::Interact:
116         this→state = std::shared_ptr<CharacterState>(new InteractState());
117         break;
118       case CharacterStateID::Move:
119         this→state = std::shared_ptr<CharacterState>(new MoveState());
120         break;
121       case CharacterStateID::Meditate:
122         this→state = std::shared_ptr<CharacterState>(new MeditateState());
123         this→animation = std::shared_ptr<Animation>(new MeditateAnimation(this
    →manager,mixer.getEffect(MusicID::Meditation)));
124         break;
125       case CharacterStateID::Attack:
126         this→state = std::shared_ptr<CharacterState>(new AttackState());
127         break;
128       case CharacterStateID::Resurrect:
129         this→state = std::shared_ptr<CharacterState>(new ResurrectState());
```



```
130         break;
131     }
132   }
133 }
134
135 CharacterStateID& NPC::getState() const {
136   return this→state→getState();
137 }
138
139 NPC::~NPC()= default;
```

jul 21, 20 15:20 **MusicManager.h** Page 1/1

```
#ifndef MUSICMANAGER_H
#define MUSICMANAGER_H
#include "Music.h"
#include "Effect.h"
#include "MusicID.h"
#include <unordered_map>
#include <string>

class MusicManager {
private:
  std::unordered_map<MusicID, Music, std::hash<MusicID>> songs;
  std::unordered_map<MusicID, Effect, std::hash<MusicID>> effects;

  void createMusic(MusicID id, const std::string& path);
  void createEffect(MusicID id, const std::string& path);
public:
  MusicManager();

  void dropMusic(MusicID id);
  void dropEffect(MusicID id);

  const Music& getMusic(MusicID id) const;

  const Effect& getEffect(MusicID id) const ;

  void loadSounds();

  ~MusicManager();

  MusicManager(const MusicManager&) = delete;
    MusicManager &operator=(const MusicManager&) = delete;
};

#endif
```

jul 21, 20 15:20 **MusicManager.cpp** Page 1/1

```
#include "MusicManager.h"
#include <utility>
#include "MusicID.h"
#include "../common/Identificators.h"

MusicManager::MusicManager() : songs(),effects() {}


void MusicManager::createMusic(MusicID id, const std::string& path) {
  Music newSong(path);
  this→songs.insert(std::make_pair(id, std::move(newSong)));
}

void MusicManager::createEffect(MusicID id, const std::string& path) {
  Effect newEffect(path);
  this→effects.insert(std::make_pair(id, std::move(newEffect)));
}

void MusicManager::dropMusic(MusicID id) {
  this→songs.erase(id);
}

void MusicManager::dropEffect(MusicID id) {
  this→effects.erase(id);
}

const Music& MusicManager::getMusic(MusicID id) const {
  return this→songs.at(id);
}

const Effect& MusicManager::getEffect(MusicID id) const {
  return this→effects.at(id);
}

void MusicManager::loadSounds() {
  std::string path(ROOT_DIR);
  createMusic(MusicID::BackGround, path + "/assets/sound/Musica Inicio.mp3");
  createEffect(MusicID::Eagle, path + "/assets/sound/Aguila.wav");
  createEffect(MusicID::Walk, path + "/assets/sound/Caminata.wav");
  createEffect(MusicID::Cure, path + "/assets/sound/Curar.wav");
  createEffect(MusicID::Sword, path + "/assets/sound/Espada.wav");
  createEffect(MusicID::Skeleton, path + "/assets/sound/Esqueleto.wav");
  createEffect(MusicID::Explotion, path + "/assets/sound/Explosion.wav");
  createEffect(MusicID::MagicArrow, path + "/assets/sound/Flecha Magica.wav");
  createEffect(MusicID::Arrow, path + "/assets/sound/Flecha.wav");
  createEffect(MusicID::Goblin, path + "/assets/sound/Goblin.wav");
  createEffect(MusicID::Ax, path + "/assets/sound/Hacha.wav");
  createEffect(MusicID::Hammer, path + "/assets/sound/Martillo.wav");
  createEffect(MusicID::Meditation, path + "/assets/sound/Meditar.wav");
  createEffect(MusicID::Misil, path + "/assets/sound/Misil.wav");
  createEffect(MusicID::Death, path + "/assets/sound/Muerte.wav");
  createEffect(MusicID::Wind, path + "/assets/sound/Viento.wav");
  createEffect(MusicID::Zombie, path + "/assets/sound/Zombie.wav");
  createEffect(MusicID::Wolf, path + "/assets/sound/Lobo.wav");
  createEffect(MusicID::Raven, path + "/assets/sound/Cuervo.wav");
  createEffect(MusicID::Heart, path + "/assets/sound/Latidos.wav");
  createEffect(MusicID::Merchant, path + "/assets/sound/Comerciante.wav");
  createEffect(MusicID::Banker, path + "/assets/sound/Banquero.wav");
  createEffect(MusicID::Priest, path + "/assets/sound/Sacerdote.wav");
}

MusicManager::~MusicManager() {}
```

jul 21, 20 15:20 **MusicID.h** Page 1/1

```
#ifndef MUSICID_H
#define MUSICID_H

enum class MusicID {
  Nothing,
  BackGround,
  Eagle,
  Wind,
  Wolf,
  Walk,
  Cure,
  Sword,
  Skeleton,
  Explotion,
  MagicArrow,
  Arrow,
  Goblin,
  Ax,
  Hammer,
  Meditation,
  Misil,
  Death,
  Zombie,
  Raven,
  Heart,
  Banker,
  Merchant,
  Priest,
};

#endif
```

jul 21, 20 15:20 **Music.h** Page 1/1

```
#ifndef MUSIC_H
#define MUSIC_H
#include <string>

typedef struct _Mix_Music Mix_Music;

class Music {
private:
  Mix_Music* music = nullptr;
  bool reproduce;
  Mix_Music* loadMusic(const std::string &fileName);

public:
  explicit Music(const std::string &fileName);
  Music(Music∧ other);

  int playMusic(int times) const ;

  void pauseMusic() const;

  void setVolume(int volume) const;

  void stopMusic() const;

  ~Music();
};

#endif
```

jul 21, 20 15:20 **Music.cpp** Page 1/1

```cpp
#include "Music.h"
#include <SDL2/SDL_mixer.h>
#include "../common/Exception.h"


Music::Music(const std::string& fileName) : reproduce(false) {
  this→music = loadMusic(fileName);
}

int Music::playMusic(int times) const {
  return Mix_PlayMusic(this→music, times);
}

Music::Music(Music ∧other) {
    std::swap(this→music, other.music);
}


Mix_Music* Music::loadMusic(const std::string &fileName) {
  if (¬this→music) {
    this→music = Mix_LoadMUS(fileName.c_str());
  }
  if (¬this→music){
    throw Exception("Fail loading music: %s",Mix_GetError());
  }
  return this→music;
}

void Music::setVolume(int volume) const{
  Mix_VolumeMusic(volume);
}

void Music::pauseMusic() const {
  if (Mix_PausedMusic() ≡ 1)
    Mix_ResumeMusic();
  else
    Mix_PauseMusic();
}

void Music::stopMusic() const {
  Mix_HaltMusic();
}

Music::~Music() {
  if (this→music) {
    Mix_FreeMusic(this→music);
    this→music = nullptr;
  }
}
```

jul 21, 20 15:20 **ZombieHead.h** Page 1/1

```cpp
#ifndef ZOMBIEHEAD_H
#define ZOMBIEHEAD_H

#include "Head.h"
#include "../TextureManager.h"

class ZombieHead: public Head {
public:
    ZombieHead(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~ZombieHead();

};

#endif
```

jul 21, 20 15:20 **ZombieHead.cpp** Page 1/1

```cpp
#include "ZombieHead.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

ZombieHead::ZombieHead(const TextureManager& manager) :
 Head(manager.getTexture(TextureID::ZombieHead),WIDTH_HEAD,HEIGHT_HEAD, HeadID::Zombie) {}

void ZombieHead::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*direction, this→height*0, this→width, this→height};
  SDL_Rect dstHead = {posX, posY+this→height−2, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void ZombieHead::update(int dir){
    this→direction = dir;
}

ZombieHead::~ZombieHead(){}
```

jul 21, 20 15:20 **ZombieBody.h** Page 1/1

```cpp
#ifndef ZOMBIEBODY_H
#define ZOMBIEBODY_H

#include "Body.h"
#include "../TextureManager.h"

class ZombieBody: public Body {
private:
    void setDirection(int direction);
public:
    ZombieBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~ZombieBody();

};

#endif
```

jul 21, 20 15:20 **ZombieBody.cpp** Page 1/1

```cpp
#include "ZombieBody.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

ZombieBody::ZombieBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::ZombieBody),WIDTH_BODY,HEIGHT_BODY,BodyID::Zombie) {}

void ZombieBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void ZombieBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void ZombieBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

ZombieBody::~ZombieBody(){}
```

jul 21, 20 15:20 **Weapon.h** Page 1/1

```cpp
#ifndef WEAPON_H
#define WEAPON_H

#include "Item.h"
#include "../Texture.h"
#include "../../common/Identificators.h"

class Weapon: public Item {
protected:
    int frame{0};
    float elapsed{0.0};
    float animationSpeed{30.0f};
    int totalFrames{5};
    WeaponID id{WeaponID::Nothing};
    Direction direction{Direction::down};
public:
    Weapon(const Texture& texture, const int width, const int height, WeaponID id = WeaponID::Nothing);

    virtual void render(int posX, int posY) = 0;

    void update(double dt, Direction dir);

    void setAnimationSpeed(float speed);

    int getHeight() const;

    WeaponID getId() const;

    ~Weapon();
};

#endif
```

jul 21, 20 15:20 **Weapon.cpp** Page 1/1

```cpp
#include "Weapon.h"

Weapon::Weapon(const Texture& texture, const int width, const int height, WeaponID id) :
        Item(texture, width, height), id(id) {}

void Weapon::update(double dt, Direction dir) {
        if (this→direction ≠ dir) {
            this→direction = dir;
            this→elapsed = 0;
        }
        this→elapsed += dt;
        this→frame = int(this→elapsed/this→animationSpeed) % this→totalFrames;
}

void Weapon::setAnimationSpeed(float speed) {
        this→animationSpeed = speed;
}

int Weapon::getHeight() const {
        return this→height;
}

WeaponID Weapon::getId() const {
        return this→id;
}

Weapon::~Weapon() {}
```

jul 21, 20 15:20 **TurtleShield.h** Page 1/1

```cpp
#ifndef TURTLESHIELD_H
#define TURTLESHIELD_H

#include "Shield.h"
#include "../TextureManager.h"

class TurtleShield: public Shield {
private:
    void setDirection(int direction);
public:
    TurtleShield(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~TurtleShield();

};

#endif
```

jul 21, 20 15:20 **TurtleShield.cpp** Page 1/1

```cpp
#include "TurtleShield.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

TurtleShield::TurtleShield(const TextureManager& manager) :
 Shield(manager.getTexture(TextureID::TurtleShield),WIDTH_BODY,HEIGHT_BODY,Shiel
dID::TurtleShield) {}

void TurtleShield::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→wid
th, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void TurtleShield::update(double dt,Direction dir) {
    setDirection(int(dir));
    Shield::update(dt,dir);
}

void TurtleShield::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

TurtleShield::~TurtleShield(){}
```

jul 21, 20 15:20 **SpiderBody.h** Page 1/1

```cpp
#ifndef SPIDERBODY_H
#define SPIDERBODY_H

#include "Body.h"
#include "../TextureManager.h"

class SpiderBody: public Body {
public:
    SpiderBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~SpiderBody();

};

#endif
```

jul 21, 20 15:20 **SpiderBody.cpp** Page 1/1

```cpp
#include "SpiderBody.h"

#define WIDTH_BODY 53
#define HEIGHT_BODY 35

SpiderBody::SpiderBody(const TextureManager& manager):
    Body(manager.getTexture(TextureID::Spider),WIDTH_BODY,HEIGHT_BODY,BodyID::Spider) {
        this→totalFrames = 4;
    }

void SpiderBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void SpiderBody::update(double dt,Direction dir) {
    Body::update(dt,dir);
}

SpiderBody::~SpiderBody(){}
```

jul 21, 20 15:20 **SkeletonBody.h** Page 1/1

```cpp
#ifndef SKELETONBODY_H
#define SKELETONBODY_H

#include "Body.h"
#include "../TextureManager.h"

class SkeletonBody: public Body {
private:
    void setDirection(int direction);
public:
    SkeletonBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~SkeletonBody();

};

#endif
```

jul 21, 20 15:20 **SkeletonBody.cpp** Page 1/1

```cpp
#include "SkeletonBody.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 52

SkeletonBody::SkeletonBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::Skeleton),WIDTH_BODY,HEIGHT_BODY,BodyID::Skeleton) {}

void SkeletonBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, int(this→width*1.2), int(this→height*1.2)};
    this→texture.render(srcBody, dstBody);
}

void SkeletonBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void SkeletonBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

SkeletonBody::~SkeletonBody(){}
```

jul 21, 20 15:20 **SimpleArc.h** Page 1/1

```cpp
#ifndef SIMPLEARC_H
#define SIMPLEARC_H

#include "Weapon.h"
#include "../TextureManager.h"

class SimpleArc: public Weapon {
private:
    void setDirection(int direction);
public:
    SimpleArc(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~SimpleArc();

};

#endif
```

jul 21, 20 15:20 **SimpleArc.cpp** Page 1/1

```cpp
#include "SimpleArc.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

SimpleArc::SimpleArc(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::SimpleArc),WIDTH_BODY,HEIGHT_BODY, WeaponID::SimpleArc) {}

void SimpleArc::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void SimpleArc::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void SimpleArc::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


SimpleArc::~SimpleArc(){}
```

jul 21, 20 15:20 **Shield.h** Page 1/1

```cpp
#ifndef SHIELD_H
#define SHIELD_H

#include "Item.h"
#include "../Texture.h"
#include "../../common/Identificators.h"

class Shield: public Item {
protected:
    int frame{0};
    float elapsed{0.0};
    float animationSpeed{30.0f};
    int totalFrames{5};
    ShieldID id{ShieldID::Nothing};
    Direction direction{Direction::down};
public:
    Shield(const Texture& texture, const int width, const int height, ShieldID id = ShieldID::Nothing);

    virtual void render(int posX, int posY) = 0;

    void update(double dt, Direction dir);

    void setAnimationSpeed(float speed);

    ShieldID getId() const;

    ~Shield();
};


#endif
```

jul 21, 20 15:20 **Shield.cpp** Page 1/1

```cpp
#include "Shield.h"

Shield::Shield(const Texture& texture, const int width, const int height, Shield
ID id) :
        Item(texture, width, height), id(id){}

void Shield::update(double dt, Direction dir) {
        if (this→direction ≠ dir) {
            this→direction = dir;
            this→elapsed = 0;
        }
        this→elapsed += dt;
        this→frame = int(this→elapsed/this→animationSpeed) % this→totalFrames
;
}

void Shield::setAnimationSpeed(float speed) {
        this→animationSpeed = speed;
}

ShieldID Shield::getId() const{
        return this→id;
}

Shield::~Shield() {}
```

jul 21, 20 15:20 **RedCommonBody.h** Page 1/1

```cpp
#ifndef REDCOMMONBODY_H
#define REDCOMMONBODY_H

#include "Body.h"
#include "../TextureManager.h"

class RedCommonBody: public Body {
private:
    void setDirection(int direction);
public:
    RedCommonBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~RedCommonBody();

};

#endif
```

jul 21, 20 15:20 **RedCommonBody.cpp** Page 1/1

```cpp
#include "RedCommonBody.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

RedCommonBody::RedCommonBody(const TextureManager& manager):
    Body(manager.getTexture(TextureID::RedCommonBody),WIDTH_BODY,HEIGHT_BODY,BodyID::RedCommon) {}

void RedCommonBody::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void RedCommonBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void RedCommonBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


RedCommonBody::~RedCommonBody(){}
```

jul 21, 20 15:20 **PriestHead.h** Page 1/1

```cpp
#ifndef PRIESTHEAD_H
#define PRIESTHEAD_H

#include "Head.h"
#include "../TextureManager.h"

class PriestHead: public Head {
public:
    PriestHead(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~PriestHead();

};

#endif
```

jul 21, 20 15:20 **PriestHead.cpp** Page 1/1

```cpp
#include "PriestHead.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 17

PriestHead::PriestHead(const TextureManager& manager) :
 Head(manager.getTexture(TextureID::PriestHead),WIDTH_HEAD,HEIGHT_HEAD,HeadID::Priest) {}

void PriestHead::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width * direction, this→height * 0, this→width, this→height};
  SDL_Rect dstHead = {posX, posY+this→height−2, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void PriestHead::update(int dir){
    this→direction = dir;
}

PriestHead::~PriestHead(){}
```

jul 21, 20 15:20 **PriestBody.h** Page 1/1

```cpp
#ifndef PRIESTBODY_H
#define PRIESTBODY_H

#include "Body.h"
#include "../TextureManager.h"

class PriestBody: public Body {
private:
    void setDirection(int direction);
public:
    PriestBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~PriestBody();

};

#endif
```

jul 21, 20 15:20 **PriestBody.cpp** Page 1/1

```cpp
#include "PriestBody.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

PriestBody::PriestBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::PriestBody),WIDTH_BODY,HEIGHT_BODY,BodyID::Priest) {}

void PriestBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void PriestBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void PriestBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

PriestBody::~PriestBody(){}
```

jul 21, 20 15:20 **PlateArmor.h** Page 1/1

```cpp
#ifndef PLATEARMOR_H
#define PLATEARMOR_H

#include "Body.h"
#include "../TextureManager.h"

class PlateArmor: public Body {
private:
    void setDirection(int direction);
public:
    PlateArmor(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~PlateArmor();

};

#endif
```

jul 21, 20 15:20 **PlateArmor.cpp** Page 1/1

```cpp
#include "PlateArmor.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

PlateArmor::PlateArmor(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::PlateArmor),WIDTH_BODY,HEIGHT_BODY,BodyID::PlateArmor) {}

void PlateArmor::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void PlateArmor::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void PlateArmor::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


PlateArmor::~PlateArmor(){}
```

jul 21, 20 15:20 **MissileAnimation.h** Page 1/1

```cpp
#ifndef MISSILEANIMATION_H
#define MISSILEANIMATION_H

#include "Animation.h"
#include "../TextureManager.h"
#include "../Effect.h"

class MissileAnimation: public Animation {
public:
    MissileAnimation(const TextureManager& manager,const Effect& effect);

    virtual void render(int x, int y);

    ~MissileAnimation();

};

#endif
```

jul 21, 20 15:20 **MissileAnimation.cpp** Page 1/1

```cpp
#include "MissileAnimation.h"

#define WIDTHFRAME 85
#define HEIGHTFRAME 70
#define FRAMES 7

MissileAnimation::MissileAnimation(const TextureManager& manager,const Effect& effect) :
 Animation(manager.getTexture(TextureID::MissileEffect),WIDTHFRAME, HEIGHTFRAME, effect,FRAMES) {}

void MissileAnimation::render(int x, int y) {
    if (this→finish)
        return;
    if(¬this→effectPlayed) {
        this→effect.playEffect(0,64);
        this→effectPlayed = true;
    }
    SDL_Rect src = {this→width*this→frame,this→height*0, this→width, this→height};
  SDL_Rect dst = {x−int(this→width/3), y−int(this→height/3), this→width, this→height};
    this→texture.render(src, dst);
}

MissileAnimation::~MissileAnimation() {}
```

jul 21, 20 15:20 **MerchantBody.h** Page 1/1

```cpp
#ifndef MERCHANTBODY_H
#define MERCHANTBODY_H

#include "Body.h"
#include "../TextureManager.h"

class MerchantBody: public Body {
private:
    void setDirection(int direction);
public:
    MerchantBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~MerchantBody();

};

#endif
```

jul 21, 20 15:20 **MerchantBody.cpp** Page 1/1

```cpp
#include "MerchantBody.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

MerchantBody::MerchantBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::Merchant),WIDTH_BODY,HEIGHT_BODY, BodyID::Merchant) {}

void MerchantBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void MerchantBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}
void MerchantBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

MerchantBody::~MerchantBody(){}
```

jul 21, 20 15:20 **MeditateAnimation.h** Page 1/1

```cpp
#ifndef MEDITATEANIMATION_H
#define MEDITATEANIMATION_H

#include "Animation.h"
#include "../TextureManager.h"
#include "../Effect.h"

class MeditateAnimation: public Animation {
public:
    MeditateAnimation(const TextureManager& manager,const Effect& effect);

    virtual void render(int x, int y);

    ~MeditateAnimation();

};

#endif
```

jul 21, 20 15:20 **MeditateAnimation.cpp** Page 1/1

```cpp
#include "MeditateAnimation.h"

#define WIDTHFRAME 80
#define HEIGHTFRAME 170
#define FRAMES 10

MeditateAnimation::MeditateAnimation(const TextureManager& manager,const Effect&
 effect) :
 Animation(manager.getTexture(TextureID::MeditateEffect),WIDTHFRAME, HEIGHTFRAME
,effect,FRAMES) {}

void MeditateAnimation::render(int x, int y) {
    if(¬this→effectPlayed) {
        this→effect.playEffect(0,64);
        this→effectPlayed = true;
    }
    int row = this→totalFrames/2;
    SDL_Rect src = {this→width*(this→frame % row),
                    this→height*int(floor(this→frame/row)), this→width, this
→height};
  SDL_Rect dst = {x−int(this→width/3), y−int(this→height/1.5), this→width, thi
s→height};
    this→texture.render(src, dst);
    this→finish=false;
}

MeditateAnimation::~MeditateAnimation() {}
```

jul 21, 20 15:20 **MagicHat.h** Page 1/1

```cpp
#ifndef MAGICHAT_H
#define MAGICHAT_H

#include "Helmet.h"
#include "../TextureManager.h"

class MagicHat: public Helmet {
public:
    MagicHat(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~MagicHat();

};

#endif
```

jul 21, 20 15:20 **MagicHat.cpp** Page 1/1

```cpp
#include "MagicHat.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

MagicHat::MagicHat(const TextureManager& manager) :
 Helmet(manager.getTexture(TextureID::MagicHat),WIDTH_HEAD,HEIGHT_HEAD,HelmetID:
:MagicHat) {}

void MagicHat::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*this→direction, this→height*0, this→width,
 this→height};
  SDL_Rect dstHead = {posX, posY−this→height/4+6, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void MagicHat::update(int dir) {
    this→direction = dir;
}

MagicHat::~MagicHat(){}
```

jul 21, 20 15:20 **MagicArrowAnimation.h** Page 1/1

```cpp
#ifndef MAGICARROWANIMATION_H
#define MAGICARROWANIMATION_H

#include "Animation.h"
#include "../TextureManager.h"
#include "../Effect.h"

class MagicArrowAnimation: public Animation {
public:
    MagicArrowAnimation(const TextureManager& manager,const Effect& effect);

    virtual void render(int x, int y);

    ~MagicArrowAnimation();

};

#endif
```

jul 21, 20 15:20 **MagicArrowAnimation.cpp** Page 1/1

```cpp
#include "MagicArrowAnimation.h"

#define WIDTHFRAME 62
#define HEIGHTFRAME 60
#define FRAMES 5

MagicArrowAnimation::MagicArrowAnimation(const TextureManager& manager,const Effect& effect) :
 Animation(manager.getTexture(TextureID::MagicArrowEffect),WIDTHFRAME, HEIGHTFRAME,effect,FRAMES) {}

void MagicArrowAnimation::render(int x, int y) {
    if (this→finish)
        return;
    if(¬this→effectPlayed) {
        this→effect.playEffect(0,64);
        this→effectPlayed = true;
    }
    SDL_Rect src = {this→width*this→frame, this→height*0, this→width, this→height};
  SDL_Rect dst = {x−int(this→width/3), y−int(this→height/3), this→width, this→height};
    this→texture.render(src, dst);
}

MagicArrowAnimation::~MagicArrowAnimation() {}
```

jul 21, 20 15:20 **LongSword.h** Page 1/1

```cpp
#ifndef LONGSWORD_H
#define LONGSWORD_H

#include "Weapon.h"
#include "../TextureManager.h"

class LongSword: public Weapon {
private:
    void setDirection(int direction);
public:
    LongSword(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~LongSword();

};

#endif
```

jul 21, 20 15:20 **LongSword.cpp** Page 1/1

```cpp
#include "LongSword.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

LongSword::LongSword(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::LongSword),WIDTH_BODY,HEIGHT_BODY, WeaponID::LongSword) {}

void LongSword::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void LongSword::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void LongSword::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


LongSword::~LongSword(){}
```

jul 21, 20 15:20 **LeatherArmor.h** Page 1/1

```cpp
#ifndef LEATHERARMOR_H
#define LEATHERARMOR_H

#include "Body.h"
#include "../TextureManager.h"

class LeatherArmor: public Body {
private:
    void setDirection(int direction);
public:
    LeatherArmor(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~LeatherArmor();

};

#endif
```

jul 21, 20 15:20 **LeatherArmor.cpp** Page 1/1

```cpp
#include "LeatherArmor.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

LeatherArmor::LeatherArmor(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::LeatherArmor),WIDTH_BODY,HEIGHT_BODY,BodyID:
:LeatherArmor) {}

void LeatherArmor::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→wid
th, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void LeatherArmor::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void LeatherArmor::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

LeatherArmor::~LeatherArmor(){}
```

jul 21, 20 15:20 **Item.h** Page 1/1

```cpp
#ifndef ITEM_H
#define ITEM_H

#include "../Texture.h"

class Item {
protected:
    const Texture& texture;
    const int width; //Ancho del Frame
    const int height; //Alto del Frame
public:
    Item(const Texture& texture, const int width, const int height);

    virtual void render(int posX, int posY);

    ~Item();
};

#endif
```

jul 21, 20 15:20 **Item.cpp** Page 1/1

```cpp
#include "Item.h"

Item::Item(const Texture& texture, const int width, const int height) :
    texture(texture), width(width), height(height) {}

void Item::render(int posX, int posY) {
        SDL_Rect srcItem = {0,0, this→width, this→height};
       SDL_Rect dstItem = {posX, posY, this→width, this→height};
         this→texture.render(srcItem, dstItem);
}

Item::~Item() {}
```

jul 21, 20 15:20 **IronShield.h** Page 1/1

```cpp
#ifndef IRONSHIELD_H
#define IRONSHIELD_H

#include "Shield.h"
#include "../TextureManager.h"

class IronShield: public Shield {
private:
    void setDirection(int direction);
public:
    IronShield(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~IronShield();

};

#endif
```

jul 21, 20 15:20 **IronShield.cpp** Page 1/1

```cpp
#include "IronShield.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

IronShield::IronShield(const TextureManager& manager) :
 Shield(manager.getTexture(TextureID::IronShield),WIDTH_BODY,HEIGHT_BODY, Shield
ID::IronShield) {}

void IronShield::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→wid
th, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void IronShield::update(double dt,Direction dir) {
    setDirection(int(dir));
    Shield::update(dt,dir);
}

void IronShield::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

IronShield::~IronShield(){}
```

jul 21, 20 15:20 **IronHelmet.h** Page 1/1

```cpp
#ifndef IRONHELMET_H
#define IRONHELMET_H

#include "Helmet.h"
#include "../TextureManager.h"

class IronHelmet: public Helmet {
public:
    IronHelmet(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~IronHelmet();

};

#endif
```

jul 21, 20 15:20 **IronHelmet.cpp** Page 1/1

```cpp
#include "IronHelmet.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

IronHelmet::IronHelmet(const TextureManager& manager) :
 Helmet(manager.getTexture(TextureID::IronHelmet),WIDTH_HEAD,HEIGHT_HEAD, HelmetID::IronHelmet) {}

void IronHelmet::render(int posX, int posY) {
    SDL_Rect srcHead = {this->width*this->direction, this->height*0, this->width, this->height};
  SDL_Rect dstHead = {posX, posY+this->height-2, this->width, this->height};
    this->texture.render(srcHead, dstHead);
}

void IronHelmet::update(int dir) {
    this->direction = dir;
}

IronHelmet::~IronHelmet(){}
```

jul 21, 20 15:20 **HumanHead.h** Page 1/1

```cpp
#ifndef HUMANHEAD_H
#define HUMANHEAD_H

#include "Head.h"
#include "../TextureManager.h"

class HumanHead: public Head {
public:
    HumanHead(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~HumanHead();

};

#endif
```

jul 21, 20 15:20 **HumanHead.cpp** Page 1/1

```cpp
#include "HumanHead.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

HumanHead::HumanHead(const TextureManager& manager) :
 Head(manager.getTexture(TextureID::HumanHead),WIDTH_HEAD,HEIGHT_HEAD,HeadID::Human) {}

void HumanHead::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*this→direction, this→height*0, this→width, this→height};
  SDL_Rect dstHead = {posX, posY+this→height−2, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void HumanHead::update(int dir){
    this→direction = dir;
}


HumanHead::~HumanHead(){}
```

jul 21, 20 15:20 **Hood.h** Page 1/1

```cpp
#ifndef HOOD_H
#define HOOD_H

#include "Helmet.h"
#include "../TextureManager.h"

class Hood: public Helmet {
public:
    Hood(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~Hood();

};

#endif
```

jul 21, 20 15:20 **Hood.cpp** Page 1/1

```cpp
#include "Hood.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

Hood::Hood(const TextureManager& manager) :
 Helmet(manager.getTexture(TextureID::Hood),WIDTH_HEAD,HEIGHT_HEAD, HelmetID::Hood) {}

void Hood::render(int posX, int posY) {
    SDL_Rect srcHead = {this->width*this->direction, this->height*0, this->width, this->height};
  SDL_Rect dstHead = {posX, posY+this->height-2, this->width, this->height};
    this->texture.render(srcHead, dstHead);
}

void Hood::update(int dir) {
    this->direction = dir;
}

Hood::~Hood(){}
```

jul 21, 20 15:20 **HitAnimation.h** Page 1/1

```cpp
#ifndef HITANIMATION_H
#define HITANIMATION_H

#include "Animation.h"
#include "../TextureManager.h"
#include "../Effect.h"

class HitAnimation: public Animation {
public:
    HitAnimation(const TextureManager& manager,const Effect& effect);

    virtual void render(int x, int y);

    ~HitAnimation();

};

#endif
```

jul 21, 20 15:20 **HitAnimation.cpp** Page 1/1

```cpp
#include "HitAnimation.h"
#include "../../common/Random.h"
#define WIDTHFRAME 15
#define HEIGHTFRAME 15
#define FRAMES 10

HitAnimation::HitAnimation(const TextureManager& manager,const Effect& effect) :
 Animation(manager.getTexture(TextureID::HitEffect),WIDTHFRAME, HEIGHTFRAME,effect,FRAMES) {}

void HitAnimation::render(int x, int y) {
    if (this→finish)
        return;
    if(¬this→effectPlayed) {
        this→effect.playEffect();
        this→effectPlayed = true;
    }
    int randX = Random::get(0,this→width);
    int randY = Random::get(0,this→height*2);
    SDL_Rect src = {this→width*this→frame, this→height*0, this→width, this→height};
  SDL_Rect dst = {x+randX, y+randY, this→width, this→height};
    this→texture.render(src, dst);
}

HitAnimation::~HitAnimation() {}
```

jul 21, 20 15:20 **Helmet.h** Page 1/1

```cpp
#ifndef HELMET_H
#define HELMET_H

#include "Item.h"
#include "../Texture.h"
#include "../../common/Identificators.h"

class Helmet: public Item {
protected:
    HelmetID id{HelmetID::Nothing};
    int direction{0};
public:
    Helmet(const Texture& texture, const int width, const int height,HelmetID id=HelmetID::Nothing) :
          Item(texture, width, height),id(id){}

    virtual void render(int posX, int posY) = 0;

    virtual void update(int dir) = 0;

    HelmetID getId(){
        return this→id;
    }

    ~Helmet() {};
};

#endif
```

jul 21, 20 15:20 **Head.h** Page 1/1

```
1  #ifndef HEAD_H
2  #define HEAD_H
3
4  #include "Item.h"
5  #include "../Texture.h"
6  #include "../../common/Identificators.h"
7
8  class Head: public Item {
9  protected:
10     HeadID id{HeadID::Nothing};
11     int direction{0};
12 public:
13     Head(const Texture& texture, const int width, const int height,HeadID id = H
   eadID::Nothing) :
14         Item(texture, width, height), id(id){}
15
16     virtual void render(int posX, int posY) = 0;
17
18     virtual void update(int dir) = 0;
19
20     HeadID getId(){
21         return this→id;
22     }
23
24     ~Head() {};
25 };
26
27 #endif
```

jul 21, 20 15:20 **Hammer.h** Page 1/1

```
1  #ifndef HAMMER_H
2  #define HAMMER_H
3
4  #include "Weapon.h"
5  #include "../TextureManager.h"
6
7  class Hammer: public Weapon {
8  private:
9      void setDirection(int direction);
10 public:
11     Hammer(const TextureManager& manager);
12
13     virtual void render(int posX, int posY);
14
15     virtual void update(double dt,Direction dir);
16
17     ~Hammer();
18
19 };
20
21 #endif
```

jul 21, 20 15:20 **Hammer.cpp** Page 1/1

```cpp
#include "Hammer.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

Hammer::Hammer(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::Hammer),WIDTH_BODY,HEIGHT_BODY, WeaponID::Hammer) {}

void Hammer::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void Hammer::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void Hammer::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


Hammer::~Hammer(){}
```

jul 21, 20 15:20 **GreenCommonBody.h** Page 1/1

```cpp
#ifndef GREENCOMMONBODY_H
#define GREENCOMMONBODY_H

#include "Body.h"
#include "../TextureManager.h"

class GreenCommonBody: public Body {
private:
    void setDirection(int direction);
public:
    GreenCommonBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~GreenCommonBody();

};

#endif
```

jul 21, 20 15:20 **GreenCommonBody.cpp** Page 1/1

```cpp
#include "GreenCommonBody.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

GreenCommonBody::GreenCommonBody(const TextureManager& manager):
    Body(manager.getTexture(TextureID::GreenCommonBody),WIDTH_BODY,HEIGHT_BODY,
BodyID::GreenCommon) {}

void GreenCommonBody::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→wid
th, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void GreenCommonBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void GreenCommonBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


GreenCommonBody::~GreenCommonBody(){}
```

jul 21, 20 15:20 **GoblinBody.h** Page 1/1

```cpp
#ifndef GOBLINBODY_h
#define GOBLINBODY_h

#include "Body.h"
#include "../TextureManager.h"

class GoblinBody: public Body {
private:
    void setDirection(int direction);
public:
    GoblinBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~GoblinBody();

};

#endif
```

jul 21, 20 15:20 **GoblinBody.cpp** Page 1/1

```cpp
#include "GoblinBody.h"

#define WIDTH_BODY 24
#define HEIGHT_BODY 31

GoblinBody::GoblinBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::Goblin),WIDTH_BODY,HEIGHT_BODY, BodyID::Goblin) {
      this→totalFrames = 8;
 }

void GoblinBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void GoblinBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void GoblinBody::setDirection(int direction) {}

GoblinBody::~GoblinBody(){}
```

jul 21, 20 15:20 **GnomeHead.h** Page 1/1

```cpp
#ifndef GNOMEHEAD_H
#define GNOMEHEAD_H

#include "Head.h"
#include "../TextureManager.h"

class GnomeHead: public Head {
public:
    GnomeHead(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~GnomeHead();

};

#endif
```

jul 21, 20 15:20 **GnomeHead.cpp** Page 1/1

```cpp
#include "GnomeHead.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

GnomeHead::GnomeHead(const TextureManager& manager) :
 Head(manager.getTexture(TextureID::GnomeHead),WIDTH_HEAD,HEIGHT_HEAD,HeadID::Gnome) {}

void GnomeHead::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*this→direction, this→height*0, this→width, this→height};
  SDL_Rect dstHead = {posX, posY+this→height−2, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void GnomeHead::update(int dir){
    this→direction = dir;
}

GnomeHead::~GnomeHead(){}
```

jul 21, 20 15:20 **GnarledStick.h** Page 1/1

```cpp
#ifndef GNARLEDSTICK_H
#define GNARLEDSTICK_H

#include "Weapon.h"
#include "../TextureManager.h"

class GnarledStick: public Weapon {
private:
    void setDirection(int direction);
public:
    GnarledStick(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~GnarledStick();

};

#endif
```

jul 21, 20 15:20 **GnarledStick.cpp** Page 1/1

```cpp
#include "GnarledStick.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

GnarledStick::GnarledStick(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::GnarledStick),WIDTH_BODY,HEIGHT_BODY, WeaponID::GnarledStick) {}

void GnarledStick::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY−2, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void GnarledStick::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void GnarledStick::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


GnarledStick::~GnarledStick(){}
```

jul 21, 20 15:20 **GhostBody.h** Page 1/1

```cpp
#ifndef GHOSTBODY_H
#define GHOSTBODY_H

#include "Body.h"
#include "../TextureManager.h"

class GhostBody: public Body {
private:
    void setDirection(int direction);
public:
    GhostBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~GhostBody();

};

#endif
```

jul 21, 20 15:20 **GhostBody.cpp** Page 1/1

```cpp
#include "GhostBody.h"

#define WIDTH_BODY 29
#define HEIGHT_BODY 32

GhostBody::GhostBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::Ghost),WIDTH_BODY,HEIGHT_BODY,BodyID::Ghost)
 {
     this→totalFrames = 3;
 }

void GhostBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void GhostBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void GhostBody::setDirection(int direction) {}

GhostBody::~GhostBody(){}
```

jul 21, 20 15:20 **ExplotionAnimation.h** Page 1/1

```cpp
#ifndef EXPLOTIONANIMATION_H
#define EXPLOTIONANIMATION_H

#include "Animation.h"
#include "../TextureManager.h"
#include "../Effect.h"

class ExplotionAnimation: public Animation {
public:
    ExplotionAnimation(const TextureManager& manager,const Effect& effect);

    virtual void render(int x, int y);

    ~ExplotionAnimation();

};

#endif
```

jul 21, 20 15:20 **ExplotionAnimation.cpp** Page 1/1

```cpp
#include "ExplotionAnimation.h"

#define WIDTHFRAME 62
#define HEIGHTFRAME 60
#define FRAMES 10

ExplotionAnimation::ExplotionAnimation(const TextureManager& manager,const Effect& effect) :
 Animation(manager.getTexture(TextureID::ExplotionEffect),WIDTHFRAME, HEIGHTFRAME,effect,FRAMES) {}

void ExplotionAnimation::render(int x, int y) {
    if (this→finish)
        return;
    if(¬this→effectPlayed) {
        this→effect.playEffect(0,64);
        this→effectPlayed = true;
    }
    int row = this→totalFrames/2;
    SDL_Rect src = {this→width*(this→frame % row),
                    this→height*int(floor(this→frame/row)), this→width, this→height};
  SDL_Rect dst = {x−int(this→width/3), y−int(this→height/3), this→width, this→height};
    this→texture.render(src, dst);
}

ExplotionAnimation::~ExplotionAnimation() {}
```

jul 21, 20 15:20 **ElficFlaute.h** Page 1/1

```cpp
#ifndef ELFICFLAUTE_H
#define ELFICFLAUTE_H

#include "Weapon.h"
#include "../TextureManager.h"

class ElficFlaute: public Weapon {
private:
    void setDirection(int direction);
public:
    ElficFlaute(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~ElficFlaute();

};

#endif
```

jul 21, 20 15:20 **ElficFlaute.cpp** Page 1/1

```cpp
#include "ElficFlaute.h"
#include "../TextureID.h"

#define WIDTH_BODY 42
#define HEIGHT_BODY 64

ElficFlaute::ElficFlaute(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::ElficFlaute),WIDTH_BODY,HEIGHT_BODY, WeaponID::ElficFlaute) {}

void ElficFlaute::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void ElficFlaute::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void ElficFlaute::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


ElficFlaute::~ElficFlaute(){}
```

jul 21, 20 15:20 **ElfHead.h** Page 1/1

```cpp
#ifndef ELFHEAD_H
#define ELFHEAD_H

#include "Head.h"
#include "../TextureManager.h"

class ElfHead: public Head {
public:
    ElfHead(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~ElfHead();

};

#endif
```

jul 21, 20 15:20 **ElfHead.cpp** Page 1/1

```cpp
#include "ElfHead.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 16

ElfHead::ElfHead(const TextureManager& manager) :
 Head(manager.getTexture(TextureID::ElfHead),WIDTH_HEAD,HEIGHT_HEAD,HeadID::Elf)
 {}

void ElfHead::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*this→direction, this→height*0, this→width,
 this→height};
  SDL_Rect dstHead = {posX, posY+this→height−2, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void ElfHead::update(int dir){
    this→direction = dir;
}

ElfHead::~ElfHead(){}
```

jul 21, 20 15:20 **DwarfHead.h** Page 1/1

```cpp
#ifndef DWARFHEAD_H
#define DWARFHEAD_H

#include "Head.h"
#include "../TextureManager.h"

class DwarfHead: public Head {
public:
    DwarfHead(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(int dir);

    ~DwarfHead();

};

#endif
```

jul 21, 20 15:20 **DwarfHead.cpp** Page 1/1

```cpp
#include "DwarfHead.h"
#include "../TextureID.h"

#define WIDTH_HEAD 17
#define HEIGHT_HEAD 19

DwarfHead::DwarfHead(const TextureManager& manager) :
 Head(manager.getTexture(TextureID::DwarfHead),WIDTH_HEAD,HEIGHT_HEAD,HeadID::Dwarf) {}

void DwarfHead::render(int posX, int posY) {
    SDL_Rect srcHead = {this->width*this->direction, this->height*0, this->width, this->height};
  SDL_Rect dstHead = {posX, posY+this->height-2, this->width, this->height};
    this->texture.render(srcHead, dstHead);
}

void DwarfHead::update(int dir){
    this->direction = dir;
}

DwarfHead::~DwarfHead(){}
```

jul 21, 20 15:20 **CureAnimation.h** Page 1/1

```cpp
#ifndef CUREANIMATION_H
#define CUREANIMATION_H

#include "Animation.h"
#include "../TextureManager.h"
#include "../Effect.h"

class CureAnimation: public Animation {
public:
    CureAnimation(const TextureManager& manager,const Effect& effect);

    virtual void render(int x, int y);

    ~CureAnimation();

};

#endif
```

jul 21, 20 15:20 **CureAnimation.cpp** Page 1/1

```cpp
#include "CureAnimation.h"

#define WIDTHFRAME 113
#define HEIGHTFRAME 113
#define FRAMES 20

CureAnimation::CureAnimation(const TextureManager& manager,const Effect& effect)
 :
 Animation(manager.getTexture(TextureID::HealthEffect),WIDTHFRAME, HEIGHTFRAME,effect,FRAMES) {}

void CureAnimation::render(int x, int y) {
    if (this→finish)
        return;
    if(¬this→effectPlayed) {
        this→effect.playEffect(0,64);
        this→effectPlayed = true;
    }
    int row = this→totalFrames/4;
    SDL_Rect src = {this→width*(this→frame % row),
                    this→height*int(floor(this→frame/row)), this→width, this→height};
  SDL_Rect dst = {x−int(this→width/2.5), y−int(this→height/3), this→width, this→height};
    this→texture.render(src, dst);
}

CureAnimation::~CureAnimation() {}
```

jul 21, 20 15:20 **Crosier.h** Page 1/1

```cpp
#ifndef CROSIER_H
#define CROSIER_H

#include "Weapon.h"
#include "../TextureManager.h"

class Crosier: public Weapon {
private:
    void setDirection(int direction);
public:
    Crosier(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~Crosier();

};

#endif
```

jul 21, 20 15:20 **Crosier.cpp** Page 1/1

```cpp
#include "Crosier.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

Crosier::Crosier(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::Crosier),WIDTH_BODY,HEIGHT_BODY, WeaponID::Crosier) {}

void Crosier::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void Crosier::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void Crosier::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


Crosier::~Crosier(){}
```

jul 21, 20 15:20 **CompoundArc.h** Page 1/1

```cpp
#ifndef COMPOUNDARC_H
#define COMPOUNDARC_H

#include "Weapon.h"
#include "../TextureManager.h"

class CompoundArc: public Weapon {
private:
    void setDirection(int direction);
public:
    CompoundArc(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~CompoundArc();

};

#endif
```

jul 21, 20 15:20 **CompoundArc.cpp** Page 1/1

```cpp
#include "CompoundArc.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

CompoundArc::CompoundArc(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::CompoundArc),WIDTH_BODY,HEIGHT_BODY, WeaponID::CompoundArc) {}

void CompoundArc::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void CompoundArc::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void CompoundArc::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


CompoundArc::~CompoundArc(){}
```

jul 21, 20 15:20 **Body.h** Page 1/1

```cpp
#ifndef BODY_H
#define BODY_H

#include "Item.h"
#include "../Texture.h"
#include "../../common/Identificators.h"

class Body: public Item {
protected:
    int frame{0};
    float elapsed{0.0};
    float animationSpeed{30.0};
    int totalFrames{5};
    BodyID id{BodyID::Nothing};
    Direction direction{Direction::down};
public:
    Body(const Texture& texture, const int width, const int height, BodyID id = BodyID::Nothing);

    virtual void render(int posX, int posY) = 0;

    void update(double dt, Direction dir);

    void setAnimationSpeed(float speed);

    int getHeight() const;

    BodyID getId()const;

    ~Body();
};

#endif
```

jul 21, 20 15:20 **Body.cpp** Page 1/1

```cpp
#include "Body.h"

Body::Body(const Texture& texture, const int width, const int height, BodyID id)
 :
        Item(texture, width, height), id(id) {}

 void Body::update(double dt, Direction dir) {
        if (this→direction ≠ dir) {
            this→direction = dir;
            this→elapsed = 0;
        }
        this→elapsed += dt;
        this→frame = int(this→elapsed/this→animationSpeed) % this→totalFrames
;
}

void Body::setAnimationSpeed(float speed) {
        this→animationSpeed = speed;
}

int Body::getHeight() const{
        return this→height;
}

BodyID Body::getId() const{
        return this→id;
}

Body::~Body() {}
```

jul 21, 20 15:20 **BlueTunic.h** Page 1/1

```cpp
#ifndef BLUETONIC_H
#define BLUETONIC_H

#include "Body.h"
#include "../TextureManager.h"

class BlueTunic: public Body {
private:
    void setDirection(int direction);
public:
    BlueTunic(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~BlueTunic();

};

#endif
```

jul 21, 20 15:20 **BlueTunic.cpp** Page 1/1

```cpp
#include "BlueTunic.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

BlueTunic::BlueTunic(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::BlueTunic),WIDTH_BODY,HEIGHT_BODY,BodyID::BlueTunic) {}

void BlueTunic::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void BlueTunic::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}


void BlueTunic::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

BlueTunic::~BlueTunic(){}
```

jul 21, 20 15:20 **BlueCommonBody.h** Page 1/1

```cpp
#ifndef BLUECOMMONBODY_H
#define BLUECOMMONBODY_H

#include "Body.h"
#include "../TextureManager.h"

class BlueCommonBody: public Body {
private:
    void setDirection(int direction);
public:
    BlueCommonBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~BlueCommonBody();

};

#endif
```

jul 21, 20 15:20 **BlueCommonBody.cpp** Page 1/1

```cpp
#include "BlueCommonBody.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

BlueCommonBody::BlueCommonBody(const TextureManager& manager):
    Body(manager.getTexture(TextureID::BlueCommonBody),WIDTH_BODY,HEIGHT_BODY,BodyID::BlueCommon) {}

void BlueCommonBody::render(int posX, int posY) {
    SDL_Rect srcHead = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstHead = {posX, posY, this→width, this→height};
    this→texture.render(srcHead, dstHead);
}

void BlueCommonBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void BlueCommonBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

BlueCommonBody::~BlueCommonBody(){}
```

jul 21, 20 15:20 **BankerBody.h** Page 1/1

```cpp
#ifndef BANKERBODY_H
#define BANKERBODY_H

#include "Body.h"
#include "../TextureManager.h"

class BankerBody: public Body {
private:
    void setDirection(int direction);
public:
    BankerBody(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~BankerBody();

};

#endif
```

jul 21, 20 15:20 **BankerBody.cpp** Page 1/1

```cpp
#include "BankerBody.h"
#include "../TextureID.h"
#include "../../common/Identificators.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

BankerBody::BankerBody(const TextureManager& manager) :
 Body(manager.getTexture(TextureID::Banker),WIDTH_BODY,HEIGHT_BODY,BodyID::Banker) {}

void BankerBody::render(int posX, int posY) {
    SDL_Rect srcBody = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstBody = {posX, posY, this→width, this→height};
    this→texture.render(srcBody, dstBody);
}

void BankerBody::update(double dt,Direction dir) {
    setDirection(int(dir));
    Body::update(dt,dir);
}

void BankerBody::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}

BankerBody::~BankerBody()= default;
```

jul 21, 20 15:20 **Ax.h** Page 1/1

```cpp
#ifndef AX_H
#define AX_H

#include "Weapon.h"
#include "../TextureManager.h"

class Ax: public Weapon {
private:
    void setDirection(int direction);
public:
    Ax(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~Ax();

};

#endif
```

jul 21, 20 15:20 **Ax.cpp** Page 1/1

```cpp
#include "Ax.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

Ax::Ax(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::Ax),WIDTH_BODY,HEIGHT_BODY, WeaponID::Ax) {}

void Ax::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void Ax::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void Ax::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


Ax::~Ax(){}
```

jul 21, 20 15:20 **AshStick.h** Page 1/1

```cpp
#ifndef ASHSTICK_H
#define ASHSTICK_H

#include "Weapon.h"
#include "../TextureManager.h"

class AshStick: public Weapon {
private:
    void setDirection(int direction);
public:
    AshStick(const TextureManager& manager);

    virtual void render(int posX, int posY);

    virtual void update(double dt,Direction dir);

    ~AshStick();

};

#endif
```

jul 21, 20 15:20 **AshStick.cpp** Page 1/1

```cpp
#include "AshStick.h"
#include "../TextureID.h"

#define WIDTH_BODY 25
#define HEIGHT_BODY 45

AshStick::AshStick(const TextureManager& manager):
    Weapon(manager.getTexture(TextureID::AshStick),WIDTH_BODY,HEIGHT_BODY, WeaponID::AshStick) {}

void AshStick::render(int posX, int posY) {
    SDL_Rect srcWeapon = {this→width*frame, this→height*int(direction), this→width, this→height};
  SDL_Rect dstWeapon = {posX, posY, this→width, this→height};
    this→texture.render(srcWeapon, dstWeapon);
}

void AshStick::update(double dt,Direction dir) {
    setDirection(int(dir));
    Weapon::update(dt,dir);
}

void AshStick::setDirection(int direction) {
    if (direction ≡ 0 ∨ direction ≡ 1)
        this→totalFrames = 6;
    if (direction ≡ 2∨ direction ≡ 3)
        this→totalFrames = 5;
}


AshStick::~AshStick(){}
```

jul 21, 20 15:20 **Animation.h** Page 1/1

```cpp
#ifndef ANIMATION_H
#define ANIMATION_H

#include "../Effect.h"
#include "../Texture.h"

class Animation {
protected:
    const Effect& effect;
    const Texture& texture;
    const int width; //Ancho del Frame
    const int height; //Alto del Frame
    int totalFrames{0};
    int frame{0};
    float elapsed{0.0};
    float animationSpeed{45.0f};
    bool finish{false};
    bool effectPlayed{false};
public:

    Animation(const Texture& texture, const int width, const int height, const Effect& effect,int totalFrames);

    virtual void render(int x, int y) = 0;

    void update();

    bool finished() const;

    ~Animation();

};

#endif
```

jul 21, 20 15:20 **Animation.cpp** Page 1/1

```cpp
#include "Animation.h"

Animation::Animation(const Texture& texture, const int width, const int height,
    const Effect& effect,int totalFrames) : effect(effect), texture(texture),
    width(width), height(height), totalFrames(totalFrames) {}

void Animation::update() {
        this→frame ++;
        if (this→frame ≥ this→totalFrames)
            this→finish = true;
        this→frame = this→frame % this→totalFrames;
}

bool Animation::finished() const {
        return this→finish;
}

Animation::~Animation() {}
```

jul 21, 20 15:20 **GameMap.h** Page 1/1

```cpp
#ifndef ARGENTUM_GAMEMAP_H
#define ARGENTUM_GAMEMAP_H

#include "Tile.h"
#include "../common/TiledMap.h"
#include "Texture.h"
#include "Camera.h"
#include "../common/Chrono.h"
#include "ui/UI.h"
#include <vector>
#include <map>

class GameMap {
private:
    std::vector<Tile> tiles;
    std::vector<Tile> groundTiles;
    std::map<int, Texture> tileSetMap;
    SDL_Renderer& renderer;
    uint16_t rows, colums;
    uint16_t width, height;
    void _loadTileSets(const std::vector<TileSet>&, SDL_Renderer&);
public:
    explicit GameMap(const TiledMap&, SDL_Renderer&);

    virtual ~GameMap();

    void drawHighLayers(Camera& camera);
    void drawGround(Camera& camera);

    int getMapWidth() const;
    int getMapHeight() const;

};


#endif //ARGENTUM_GAMEMAP_H
```

jul 21, 20 15:20 **GameMap.cpp** Page 1/2

```cpp
#include <iostream>
#include "GameMap.h"

GameMap::GameMap(const TiledMap & tiledMap, SDL_Renderer& renderer) : renderer(renderer) {
    this→rows = tiledMap.getHeight();
    this→colums = tiledMap.getWidth();
    this→width = tiledMap.getWidth() * tiledMap.getTileWidth();
    this→height = tiledMap.getHeight() * tiledMap.getTileHeight();
    _loadTileSets(tiledMap.getTilesets(), renderer);

    uint8_t tileWidth = tiledMap.getTileWidth();
    uint8_t tileHeight = tiledMap.getTileHeight();

    for (auto& layer : tiledMap.getTileLayers()) {

        std::vector<uint16_t> layerData = layer.getData();

        for (int y = 0; y < rows; ++y) {
            for (int x = 0; x < colums; ++x) {
                int tileIndex = x + (y * colums);
                int curGid = layerData[tileIndex];

                if (curGid == 0) {
                    continue;
                }

                int tileSetGid = -1;
                for (auto& ts : tileSetMap) {
                    if (ts.first <= curGid ) {
                        tileSetGid = ts.first;
                    }
                }

                if (tileSetGid == -1) {
                    continue;
                }
                Texture& loadTexture = tileSetMap.at(tileSetGid);

                curGid -= tileSetGid;

                int regionX = (curGid % (loadTexture.getWidth() / tileWidth)) * tileWidth;
                int regionY = (curGid / (loadTexture.getWidth() / tileHeight)) * tileHeight;

                int xPos = x * tileWidth;
                int yPos = y * tileHeight;

                Tile aTile(xPos, yPos, regionX, regionY, tileWidth, tileHeight, loadTexture);
                if (layer.isGroundLayer()) {
                    groundTiles.push_back(aTile);
                } else {
                    tiles.push_back(aTile);
                }
            }
        }
    }

}
int GameMap::getMapHeight() const {
    return this→height;
}

int GameMap::getMapWidth() const {
```

jul 21, 20 15:20 **GameMap.cpp** Page 2/2

```cpp
    return this→width;
}


void GameMap::_loadTileSets(const std::vector<TileSet> &tileSets, SDL_Renderer &renderer) {
    SDL_Color textColor = {0x0, 0x0, 0x0};
    for  (auto& aTileSet : tileSets) {
        Texture aTexture(aTileSet.getImage(), renderer, textColor);
        tileSetMap.insert(std::pair<int, Texture>(aTileSet.getFirstgid(), std::move(aTexture)));
    }
}

void GameMap::drawHighLayers(Camera& camera) {
    for (Tile& tile: tiles) {
        tile.draw(renderer, camera);
    }
}

void GameMap::drawGround(Camera& camera) {
    for (Tile& tile: groundTiles) {
        tile.draw(renderer, camera);
    }
}

GameMap::~GameMap() = default;
```

jul 21, 20 15:20 **Game.h** Page 1/1

```h
#ifndef GAME_H
#define GAME_H

#include <memory>
#include <unordered_map>
#include "Window.h"
#include "TextureManager.h"
#include "MusicManager.h"
#include "GameMap.h"
#include "Player.h"
#include "NPC.h"
#include "Camera.h"
#include "ui/UI.h"
#include "../common/InputQueue.h"
#include "../common/DataQueue.h"
#include "../common/Socket.h"
#include "../common/Identificators.h"
#include "../common/CommunicationProtocol.h"
#include "Dispatcher.h"
#include "Receiver.h"

//Clase destinada a coordinar el juego del lado del cliente
//Realiza la conexiÃ³n inicial y luego, irÃ¡ realizando los updates
//recibidos desde el server para renderizarlos y enviando
//comandos al mismo. A su vez tendrÃ¡ otros 2 Threads que
//establecerÃ¡n la comunicaciÃ³n con el server

class Game {
private:
    Window window;
    CommunicationProtocol protocol;
    TextureManager textureManager;
    MusicManager musicManager;
    std::unordered_map<uint, NPC> npcs;
    std::shared_ptr<Player> player = nullptr;
    std::shared_ptr<GameMap> map = nullptr;
    InputQueue commandQueue;
    DataQueue dataQueue;
    Dispatcher dispatcher;
    Receiver receiver;
    std::shared_ptr<Camera> camera{nullptr};
    std::shared_ptr<UI> ui{nullptr};

    RaceID translateRace(const std::string& race);
    GameClassID translateGameClass(const std::string& gameClass);
    void recieveMapAndPlayer();

    void update();
    void render();
    void sounds();
    void close();
public:
    Game();
    //Se conecta con el servidor, enviando la raza y clase elegida
    //Recibe el mapa estÃ¡tico y el primer playerInfo que tendrÃ¡ la
    //informaciÃ³n inicial para que el jugador pueda comenzar el juego.
    bool init(char* argv[]);

    //Gameloop principal del cliente.
    int run();

    ~Game();

};

#endif
```

jul 21, 20 15:20 **Game.cpp** Page 1/5

```cpp
#include "Game.h"
#include <memory>
#include <stdexcept>
#include "Presentation.h"
#include <arpa/inet.h>
#include <iostream>
#include "../common/Decoder.h"
#include "../common/SocketException.h"
#include <algorithm>
#include "../common/Random.h"

#define WRONGRACE "Raza invalida. Seleccione entre: elfo, gnomo, humano, enano."
#define WRONGCLASS "Clase invalida. Seleccione entre: mago, clerigo, paladin, guerrero"
#define ARGENTUM "Argentum Taller"
#define INITERROR "Error en Game::init: "
#define GAMELOOPTIME 1000000/45.0
#define PLAYERINFOMSG 0x01
#define OBJECTSINFOMSG 0x02
#define INTERACTMSG 0x05


Game::Game() : window(ARGENTUM), protocol(), textureManager(window.getRenderer()),
    musicManager(), npcs(), commandQueue(true), dataQueue(false),
    dispatcher(protocol,commandQueue), receiver(protocol,dataQueue) {}

void Game::recieveMapAndPlayer() {

    Message msgMap = this→protocol.receive();
    TiledMap tiledMap = Decoder::decodeMap(msgMap);

    Message msgPlayerInfo = this→protocol.receive();
    PlayerInfo info = Decoder::decodePlayerInfo(msgPlayerInfo);

    this→map = std::make_shared<GameMap>(tiledMap,this→window.getRenderer());
    this→player = std::make_shared<Player>(this→textureManager, info, this→musicManager);
}

bool Game::init(char* argv[]) {
    try{
        this→protocol.connect(argv[3], argv[4]);
        std::cout << "me conecte a " << argv[4] << std::endl;
        std::vector<uint8_t> initMsg;
        initMsg = Decoder::encodeInit(translateRace(argv[1]),translateGameClass(argv[2]));
        this→protocol.send(initMsg);
        this→textureManager.loadTextures();
        this→musicManager.loadSounds();
        recieveMapAndPlayer();

        this→camera = std::make_shared<Camera>(this→window, this→map→getMapWidth(), this→map→getMapHeight());
        this→ui = std::make_shared<UI>(this→window, &(*this→player), this→textureManager, this→musicManager);

    } catch (const SocketException& e) {
        std::cout << INITERROR << e.what() << std::endl;
        return false;
    }
    return true;
}

int Game::run() {
    const Music& musica = musicManager.getMusic(MusicID::BackGround);
  musica.playMusic(-1);
```

```
    musica.setVolume(MIX_MAX_VOLUME/4);
    Presentation presentation(window, textureManager);
  if (presentation.run())
    return 0;

    this→dispatcher.start();
    this→receiver.start();
    bool quit = false;
    SDL_Event event;

    Chrono chrono;
  double initLoop, endLoop, sleep;
  InputInfo input;
    while (¬quit) {
        try{
        initLoop = chrono.lap();
        while (SDL_PollEvent(&event) ≠ 0) {
          if (event.type ≡ SDL_QUIT) {
            quit = true;
                  break;
          }
          if(event.type ≡ SDL_KEYDOWN) {
            if (event.key.keysym.sym ≡ SDLK_m) {
              musica.pauseMusic();
            }
          }
                input = player→handleEvent(event,*camera);
                this→commandQueue.push(input);
          input = ui→handleClick(event);
          this→commandQueue.push(input);
          window.handleEvent(event);
        }

            this→update();
            this→render();
            this→sounds();
        endLoop = chrono.lap();
        sleep = GAMELOOPTIME − (endLoop − initLoop);
        if (sleep > 0)
          usleep(sleep);
      } catch (const std::exception& e) {
            quit = true;
            std::cerr << e.what() << std::endl;
        } catch (...) {
            quit = true;
            std::cerr << "Unkown Error in Game::run()" << std::endl;
        }
    }
    close();
    return 0;

}

void Game::update() {
    Message msg;
    PlayerInfo playerInfo;
    std::vector<GameObjectInfo> objects;
    std::unordered_map<uint,NPC> newNpcs;
    NPCInfo items;
    while (¬this→dataQueue.empty()){
        msg = this→dataQueue.pop();
        if (msg.getType() ≡ PLAYERINFOMSG) {
            playerInfo = Decoder::decodePlayerInfo(msg);
            this→player→updatePlayerInfo(playerInfo);
        } else if (msg.getType() ≡ OBJECTSINFOMSG) {
            objects = Decoder::decodeGameObjects(msg);
```



```
            newNpcs.clear();
            for (GameObjectInfo& npc : objects) {
                auto iter = this→npcs.find(npc.getId());
                if (iter ≡ this→npcs.end()){
                    NPC aNpc(this→textureManager,npc,this→musicManager);
                    newNpcs.insert({npc.getId(),aNpc});
                } else {
                    (*iter).second.updatePlayerInfo(npc);
                    newNpcs.insert({(*iter).first,(*iter).second});
                }
            }
            this→npcs.swap(newNpcs);
        } else if (msg.getType() ≡ INTERACTMSG) {
            items = Decoder::decodeNPCInfo(msg);
            this→ui→setNPCInfo(items);
        }
    }
    this→player→update(GAMELOOPTIME);
    for (auto& npc: this→npcs)
        npc.second.update(GAMELOOPTIME);
}


bool comparation(Character* c1, Character* c2) {
    return c1→getPosition().y < c2→getPosition().y;
}

void Game::render() {
    window.clearScreen();

    ui→render();
        Point* center = this→player→getCenter();
        camera→setPlayer(center);
      camera→render(*center);
        this→map→drawGround(*camera);

        std::vector<Character*> characters;
        std::vector<Character*> items;

        characters.push_back(&(*this→player));

    for (auto& aNPC : this→npcs) {
            if(aNPC.second.isItem()){
                items.push_back(&(aNPC.second));
            } else {
                characters.push_back(&(aNPC.second));
            }
    }

        std::sort(characters.begin(),characters.end(),comparation);
        std::sort(items.begin(),items.end(),comparation);

        for (auto& aItem : items) {
            aItem→render(*camera);
        }

        for (auto& aCharacter : characters) {
            aCharacter→render(*camera);
        }

    this→map→drawHighLayers(*camera);

    this→window.render();

}
```

jul 21, 20 15:20 **Game.cpp** Page 4/5

```
194 void Game::sounds() {
195     int random = Random::get(1,5000);
196     MusicID effectId = MusicID::Nothing;
197     switch (random) {
198         case 1:
199             effectId = MusicID::Wind;
200             break;
201         case 2:
202             effectId = MusicID::Eagle;
203             break;
204         case 3:
205             effectId = MusicID::Wolf;
206             break;
207         case 4:
208             effectId = MusicID::Raven;
209             break;
210         default:
211             break;
212     }
213     if (effectId ≠ MusicID::Nothing){
214         const Effect& effect = this→musicManager.getEffect(effectId);
215         effect.playEffect();
216     }
217 }
218
219 void Game::close() {
220     std::cout << "Cierro todo" << std::endl;
221     this→dataQueue.close();
222     this→commandQueue.close();
223     this→receiver.stop();
224     this→receiver.join();
225     this→dispatcher.stop();
226     this→dispatcher.join();
227 }
228
229 RaceID Game::translateRace(const std::string& race){
230     RaceID selectedRace;
231     if (race ≡ "elfo") {
232         selectedRace = RaceID::Elf;
233     } else if (race ≡ "enano") {
234         selectedRace = RaceID::Dwarf;
235     } else if (race ≡ "humano") {
236         selectedRace = RaceID::Human;
237     } else if (race ≡ "gnomo") {
238         selectedRace = RaceID::Gnome;
239     } else {
240         throw std::invalid_argument(WRONGRACE);
241     }
242     return selectedRace;
243 }
244
245 GameClassID Game::translateGameClass(const std::string& gameClass){
246     GameClassID selectedClass;
247     if (gameClass ≡ "mago") {
248         selectedClass = GameClassID::Mage;
249     } else if (gameClass ≡ "clerigo") {
250         selectedClass = GameClassID::Cleric;
251     } else if (gameClass ≡ "guerrero") {
252         selectedClass = GameClassID::Warrior;
253     } else if (gameClass ≡ "paladin") {
254         selectedClass = GameClassID::Paladin;
255     } else {
256         throw std::invalid_argument(WRONGCLASS);
257     }
258     return selectedClass;
259 }
```

jul 21, 20 15:20 **Game.cpp** Page 5/5

```
260
261 Game::~Game() {}
```

jul 21, 20 15:20 **Font.h** Page 1/1

```
#ifndef FONT_H
#define FONT_H
#include <SDL2/SDL_ttf.h>
#include <string>

class SDL_Texture;
class SDL_Color;
class SDL_Renderer;

class Font {
private:
  TTF_Font* font;
  int size;
  SDL_Color color;
public:
  Font(const std::string& path, const int size, SDL_Color color);

  void setColor(SDL_Color color);

  SDL_Texture* createText(const std::string& text,SDL_Renderer* renderer, int* width, int* height);

  TTF_Font* getFont() const;

  void setSize(int size);

  void  deleteText(SDL_Texture* text);

  ~Font();
};

#endif
```

jul 21, 20 15:20 **Font.cpp** Page 1/1

```
#include "Font.h"
#include "../common/Exception.h"
#include "../common/Identificators.h"

Font::Font(const std::string& path, const int size, SDL_Color color) : size(size),
  color(color) {
  std::string root(ROOT_DIR+std::string("/")+path);
  this→font = TTF_OpenFont(root.c_str(), this→size);
  if (¬this→font)
    throw Exception("Fail TTF_OpenFont: %s", TTF_GetError());
}


void Font::setColor(SDL_Color color) {
  this→color = color;
}

void Font::setSize(int size) {
  this→size = size;
}

SDL_Texture* Font::createText(const std::string& text, SDL_Renderer* renderer, int* width, int* height) {
  SDL_Surface* textSurface = TTF_RenderText_Solid(this→font, text.c_str(), this→color);
  if (¬textSurface)
    throw Exception("Fail TTF_RenderText_Solid: %s", TTF_GetError());

  SDL_Texture* texture = SDL_CreateTextureFromSurface(renderer, textSurface);
  if (¬texture)
    throw Exception("Fail SDL_CreateTextureFromSurface in createText: %s", SDL_GetError());

  *height = textSurface→h;
  *width = textSurface→w;
  SDL_FreeSurface(textSurface);
  return texture;
}

TTF_Font* Font::getFont() const {
  return this→font;
}

void Font::deleteText(SDL_Texture* text) {
  SDL_DestroyTexture(text);
}

Font::~Font(){
  if(this→font) {
    TTF_CloseFont(this→font);
    this→font = nullptr;
  }
}
```

jul 21, 20 15:20 **Effect.h** Page 1/1

```
1  #ifndef EFFECT_H
2  #define EFFECT_H
3
4  #include <SDL2/SDL_mixer.h>
5  #include <string>
6
7  class Effect {
8  private:
9    Mix_Chunk* effect = nullptr;
10     mutable int channel;
11 public:
12   explicit Effect(const std::string &fileName);
13
14   Effect(Effect∧ other);
15
16   void playEffect(int times = 0,int volume=−1) const;
17
18   void setDistance(int distance) const;
19
20   void pause() const;
21
22   ~Effect();
23 };
24
25 #endif
```

jul 21, 20 15:20 **Effect.cpp** Page 1/1

```
1  #include "Effect.h"
2  #include "../common/Exception.h"
3  #include <iostream>
4
5  Effect::Effect(const std::string& fileName) {
6    this→effect = Mix_LoadWAV(fileName.c_str());
7    if(this→effect ≡ NULL) {
8      std::cout << "ERROR al cargar el archivo" << fileName << std::endl;
9    }
10     this→channel = −1;
11 }
12
13 void Effect::playEffect(int times,int volume) const {
14   Mix_VolumeChunk(this→effect,volume);
15   this→channel = Mix_PlayChannel(−1,this→effect,times);
16 }
17
18 void Effect::pause() const {
19   Mix_Pause(this→channel);
20 }
21
22 Effect::Effect(Effect ∧other) {
23     std::swap(this→effect, other.effect);
24     std::swap(this→channel,other.channel);
25 }
26
27 void Effect::setDistance(int distance) const {
28   Mix_SetDistance(this→channel,distance);
29 }
30
31 Effect::~Effect() {
32   if (this→effect) {
33     Mix_FreeChunk(this→effect);
34     this→effect = nullptr;
35   }
36 }
```

jul 21, 20 15:20 **Dispatcher.h** Page 1/1

```h
#ifndef DISPATCHER_H
#define DISPATCHER_H

#include "../common/Thread.h"
#include "../common/InputQueue.h"
#include "../common/CommunicationProtocol.h"

//Thread encargado de enviar al servidor los comandos ingresados
//por el jugador.
class Dispatcher: public Thread {
private:
    InputQueue& queue;
    std::atomic<bool> keepTalking;
    CommunicationProtocol& protocol;
public:

    Dispatcher(CommunicationProtocol& protocol, InputQueue& queue);

    virtual void run();

    bool is_alive() const;

    void stop();

    virtual ~Dispatcher();

};

#endif
```

jul 21, 20 15:20 **Dispatcher.cpp** Page 1/1

```cpp
#include "Dispatcher.h"
#include <vector>
#include <iostream>
#include "../common/SocketException.h"
#include "../common/Decoder.h"

#define UNKNOW_ERROR "Unknow Error"
#define ERRORSOCKET "Error en la comunicaciÃ³n en Dispatcher::run() "
#define ERRORDISPATCHER "Error en Dispatcher::run() "

Dispatcher::Dispatcher(CommunicationProtocol& protocol, InputQueue& queue) :
    queue(queue), keepTalking(true), protocol(protocol) {}

void Dispatcher::run() {
    InputInfo info;
    std::vector<uint8_t> msg;
    while (this→keepTalking) {
        try{
            info = this→queue.pop();
            if (info.input ≠ InputID::nothing) {
                msg = Decoder::encodeCommand(info);
                this→protocol.send(msg);
            }
        } catch (const SocketException& e) {
            std::cerr << ERRORSOCKET << e.what() << std::endl;
            this→keepTalking = false;
        } catch(const std::exception& e){
            std::cerr << ERRORDISPATCHER << e.what() << std::endl;
            this→keepTalking = false;
        }catch (...) {
            this→keepTalking = false;
            std::cerr << UNKNOW_ERROR << std::endl;
        }

    }
}

bool Dispatcher::is_alive() const {
    return this→keepTalking;
}

void Dispatcher::stop() {
    this→keepTalking = false;
    this→protocol.stop();
}

Dispatcher::~Dispatcher() {}
```

jul 21, 20 15:20 **client_main.cpp** Page 1/1

```cpp
#include "../common/Socket.h"
#include "../common/CommunicationProtocol.h"
#include "Game.h"
#include <iostream>
#define ARGENTUM "Argentum Online"
#define INVALID_ARGUMENTS "USO: argentum_client <raza> <clase> <host> <puerto>"
#define GAMELOOPTIME 1000000/30.0

int main(int argc, char* argv[]) {
  if (argc ≠ 5) {
    std::cout << INVALID_ARGUMENTS << std::endl;
  return 1;
  }

  Game game;
  if (¬game.init(argv)) {
    return 1;
  }
  game.run();

  return 0;
}
```

jul 21, 20 15:20 **StillState.h** Page 1/1

```cpp
#ifndef STILLSTATE_H
#define STILLSTATE_H

#include "CharacterState.h"

class StillState: public CharacterState {
public:
    StillState();
    ~StillState();

    virtual InputInfo moveUp(Character& character);
    virtual InputInfo moveDown(Character& character);
    virtual InputInfo moveLeft(Character& character);
    virtual InputInfo moveRight(Character& character);
    virtual InputInfo stopMove(Character& character);

    virtual InputInfo selectItem(Character& character, int item);
    virtual InputInfo dropItem(Character& character, int item);
    virtual InputInfo unequipItem(Character& character, int item);
    virtual InputInfo buyItem(Character& character, int item);
    virtual InputInfo sellItem(Character& character, int item);

    virtual InputInfo deposit(Character& character,int item, bool isItem);
    virtual InputInfo retire(Character& character,int item, bool isItem);

    virtual InputInfo meditate(Character& character);
    virtual InputInfo resurrect(Character& character);
    virtual InputInfo cure(Character& character);
    virtual InputInfo takeItem(Character& character);
    virtual InputInfo selectTarget(Character& character, Point position);
};

#endif
```

jul 21, 20 15:20 **StillState.cpp** Page 1/3

```cpp
#include "StillState.h"

StillState::StillState() :
    CharacterState(CharacterStateID::Still){}

StillState::~StillState() {}

InputInfo StillState::moveUp(Character& character){
    InputInfo info;
    info.input = InputID::up;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo StillState::moveDown(Character& character) {
    InputInfo info;
    info.input = InputID::down;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo StillState::moveLeft(Character& character) {
    InputInfo info;
    info.input = InputID::left;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo StillState::moveRight(Character& character) {
    InputInfo info;
    info.input = InputID::right;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo StillState::stopMove(Character& character) {
    InputInfo info;
    info.input = InputID::stopMove;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo StillState::selectItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::equipItem;
    info.aditional = item;
    return info;
}


InputInfo StillState::selectTarget(Character& character, Point position) {
    InputInfo info;
    info.position = position;
    info.input =InputID::selectTarget;
    return info;
}

InputInfo StillState::meditate(Character& character) {
    InputInfo info;
```

jul 21, 20 15:20 **StillState.cpp** Page 2/3

```cpp
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::meditate;
    return info;
}

InputInfo StillState::resurrect(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::resurrect;
    return info;
}

InputInfo StillState::cure(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo StillState::takeItem(Character& character) {
    InputInfo info;
    info.position = character.getPosition();
    info.input =InputID::takeItem;
    return info;
}

InputInfo StillState::dropItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::dropItem;
    info.aditional = item;
    return info;
}

InputInfo StillState::buyItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo StillState::sellItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo StillState::retire(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::nothing;
    return info;
}

InputInfo StillState::deposit(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
```

jul 21, 20 15:20 **StillState.cpp** Page 3/3

```
133     info.input = InputID::nothing;
134     return info;
135 }
136
137 InputInfo StillState::unequipItem(Character& character, int item) {
138     InputInfo info;
139     Point aux(0.0,0.0);
140     info.position = aux;
141     info.input = InputID::unequipItem;
142     info.aditional = item;
143     return info;
144 }
```

jul 21, 20 15:20 **ResurrectState.h** Page 1/1

```
1  #ifndef RESURRECTSTATE_H
2  #define RESURRECTSTATE_H
3
4  #include "CharacterState.h"
5
6  class ResurrectState: public CharacterState {
7  public:
8      ResurrectState();
9      ~ResurrectState();
10
11     virtual InputInfo moveUp(Character& character);
12     virtual InputInfo moveDown(Character& character);
13     virtual InputInfo moveLeft(Character& character);
14     virtual InputInfo moveRight(Character& character);
15     virtual InputInfo stopMove(Character& character);
16
17     virtual InputInfo selectItem(Character& character, int item);
18     virtual InputInfo dropItem(Character& character, int item);
19     virtual InputInfo unequipItem(Character& character, int item);
20     virtual InputInfo buyItem(Character& character, int item);
21     virtual InputInfo sellItem(Character& character, int item);
22
23     virtual InputInfo deposit(Character& character,int item, bool isItem);
24     virtual InputInfo retire(Character& character,int item, bool isItem);
25
26     virtual InputInfo meditate(Character& character);
27     virtual InputInfo resurrect(Character& character);
28     virtual InputInfo cure(Character& character);
29     virtual InputInfo takeItem(Character& character);
30     virtual InputInfo selectTarget(Character& character, Point position);
31 };
32
33 #endif
```

jul 21, 20 15:20 **ResurrectState.cpp** Page 1/3

```cpp
#include "ResurrectState.h"

ResurrectState::ResurrectState() :
    CharacterState(CharacterStateID::Resurrect){}

ResurrectState::~ResurrectState() {}

InputInfo ResurrectState::moveUp(Character& character){
    InputInfo info;
    info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo ResurrectState::moveDown(Character& character) {
    InputInfo info;
    info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo ResurrectState::moveLeft(Character& character) {
    InputInfo info;
    info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo ResurrectState::moveRight(Character& character) {
    InputInfo info;
    info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo ResurrectState::stopMove(Character& character) {
    InputInfo info;
    info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo ResurrectState::selectItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::nothing;
    return info;
}

InputInfo ResurrectState::selectTarget(Character& character, Point position) {
    InputInfo info;
    info.position = position;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::meditate(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
```

jul 21, 20 15:20 **ResurrectState.cpp** Page 2/3

```cpp
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::resurrect(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::cure(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::takeItem(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::dropItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::buyItem(Character& character,int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::sellItem(Character& character,int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo ResurrectState::retire(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::nothing;
    return info;
}

InputInfo ResurrectState::deposit(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::nothing;
    return info;
```

jul 21, 20 15:20 **ResurrectState.cpp** Page 3/3

```
133 }
134
135 InputInfo ResurrectState::unequipItem(Character& character, int item) {
136     InputInfo info;
137     Point aux(0.0,0.0);
138     info.position = aux;
139     info.input = InputID::nothing;
140     return info;
141 }
142
```

jul 21, 20 15:20 **MoveState.h** Page 1/1

```
1  #ifndef MOVESTATE_H
2  #define MOVESTATE_H
3
4  #include "CharacterState.h"
5
6  class MoveState: public CharacterState {
7  public:
8      MoveState();
9      ~MoveState();
10
11     virtual InputInfo moveUp(Character& character);
12     virtual InputInfo moveDown(Character& character);
13     virtual InputInfo moveLeft(Character& character);
14     virtual InputInfo moveRight(Character& character);
15     virtual InputInfo stopMove(Character& character);
16
17     virtual InputInfo selectItem(Character& character, int item);
18     virtual InputInfo dropItem(Character& character, int item);
19     virtual InputInfo unequipItem(Character& character, int item);
20     virtual InputInfo deposit(Character& character,int item, bool isItem);
21     virtual InputInfo retire(Character& character,int item, bool isItem);
22
23     virtual InputInfo buyItem(Character& character, int item);
24     virtual InputInfo sellItem(Character& character, int item);
25
26     virtual InputInfo meditate(Character& character);
27     virtual InputInfo resurrect(Character& character);
28     virtual InputInfo cure(Character& character);
29     virtual InputInfo takeItem(Character& character);
30     virtual InputInfo selectTarget(Character& character, Point position);
31 };
32
33 #endif
```

jul 21, 20 15:20 **MoveState.cpp** Page 1/3

```cpp
#include "MoveState.h"

MoveState::MoveState() :
    CharacterState(CharacterStateID::Move){}

MoveState::~MoveState() = default;

InputInfo MoveState::moveUp(Character& character){
    InputInfo info;
    info.input = InputID::up;
    if (character.getDirection() == Direction::up)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MoveState::moveDown(Character& character) {
    InputInfo info;
    info.input = InputID::down;
    if (character.getDirection() == Direction::down)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MoveState::moveLeft(Character& character) {
    InputInfo info;
    info.input = InputID::left;
    if (character.getDirection() == Direction::left)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MoveState::moveRight(Character& character) {
    InputInfo info;
    info.input = InputID::right;
    if (character.getDirection() == Direction::right)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MoveState::stopMove(Character& character) {
    InputInfo info;
    info.input = InputID::stopMove;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MoveState::selectItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::selectTarget(Character& character, Point position) {
    InputInfo info;
    info.position = position;
```

jul 21, 20 15:20 **MoveState.cpp** Page 2/3

```cpp
    info.input =InputID::selectTarget;
    return info;
}

InputInfo MoveState::meditate(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::resurrect(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::cure(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::takeItem(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::dropItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::dropItem;
    info.aditional = item;
    return info;
}

InputInfo MoveState::buyItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::sellItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MoveState::retire(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::nothing;
```

jul 21, 20 15:20 **MoveState.cpp** Page 3/3

```
133     return info;
134 }
135
136 InputInfo MoveState::deposit(Character& character,int item, bool isItem) {
137     InputInfo info;
138     Point aux(0.0,0.0);
139     info.position = aux;
140     info.input = InputID::nothing;
141     return info;
142 }
143
144 InputInfo MoveState::unequipItem(Character& character,int item) {
145     InputInfo info;
146     Point aux(0.0,0.0);
147     info.position = aux;
148     info.input = InputID::nothing;
149     return info;
150 }
```

jul 21, 20 15:20 **MeditateState.h** Page 1/1

```
1  #ifndef MEDITATESTATE_H
2  #define MEDITATESTATE_H
3
4  #include "CharacterState.h"
5
6  class MeditateState: public CharacterState {
7  public:
8      MeditateState();
9      ~MeditateState();
10
11     virtual InputInfo moveUp(Character& character);
12     virtual InputInfo moveDown(Character& character);
13     virtual InputInfo moveLeft(Character& character);
14     virtual InputInfo moveRight(Character& character);
15     virtual InputInfo stopMove(Character& character);
16
17     virtual InputInfo selectItem(Character& character, int item);
18     virtual InputInfo dropItem(Character& character, int item);
19     virtual InputInfo unequipItem(Character& character, int item);
20     virtual InputInfo buyItem(Character& character, int item);
21     virtual InputInfo sellItem(Character& character, int item);
22
23     virtual InputInfo deposit(Character& character,int item, bool isItem);
24     virtual InputInfo retire(Character& character,int item, bool isItem);
25
26     virtual InputInfo meditate(Character& character);
27     virtual InputInfo resurrect(Character& character);
28     virtual InputInfo cure(Character& character);
29     virtual InputInfo takeItem(Character& character);
30     virtual InputInfo selectTarget(Character& character, Point position);
31 };
32
33 #endif
```

jul 21, 20 15:20 **MeditateState.cpp** Page 1/3

```
#include "MeditateState.h"

MeditateState::MeditateState() :
    CharacterState(CharacterStateID::Meditate){}

MeditateState::~MeditateState() {}

InputInfo MeditateState::moveUp(Character& character){
    InputInfo info;
    info.input = InputID::up;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MeditateState::moveDown(Character& character) {
    InputInfo info;
    info.input = InputID::down;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MeditateState::moveLeft(Character& character) {
    InputInfo info;
    info.input = InputID::left;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MeditateState::moveRight(Character& character) {
    InputInfo info;
    info.input = InputID::right;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MeditateState::stopMove(Character& character) {
    InputInfo info;
    info.input = InputID::stopMove;
    Point aux(0.0,0.0);
    info.position = aux;
    return info;
}

InputInfo MeditateState::selectItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::equipItem;
    info.aditional = item;
    return info;
}

InputInfo MeditateState::selectTarget(Character& character, Point position) {
    InputInfo info;
    info.position = position;
    info.input =InputID::selectTarget;
    return info;
}

InputInfo MeditateState::meditate(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
```

jul 21, 20 15:20 **MeditateState.cpp** Page 2/3

```
    info.position = aux;
    info.input =InputID::meditate;
    return info;
}

InputInfo MeditateState::resurrect(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MeditateState::cure(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MeditateState::takeItem(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MeditateState::dropItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::dropItem;
    info.aditional = item;
    return info;
}

InputInfo MeditateState::buyItem(Character& character,int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MeditateState::sellItem(Character& character,int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    return info;
}

InputInfo MeditateState::retire(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input = InputID::nothing;
    return info;
}

InputInfo MeditateState::deposit(Character& character,int item, bool isItem) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
```

jul 21, 20 15:20 **MeditateState.cpp** Page 3/3

```
133     info.input = InputID::nothing;
134     return info;
135 }
136
137 InputInfo MeditateState::unequipItem(Character& character, int item) {
138     InputInfo info;
139     Point aux(0.0,0.0);
140     info.position = aux;
141     info.input = InputID::unequipItem;
142     info.aditional = item;
143     return info;
144 }
145
```

jul 21, 20 15:20 **InteractState.h** Page 1/1

```
1  #ifndef INTERACTSTATE_H
2  #define INTERACTSTATE_H
3
4  #include "CharacterState.h"
5
6  class InteractState: public CharacterState {
7  private:
8      InputID beforeInput;
9  public:
10     InteractState();
11     ~InteractState();
12
13     virtual InputInfo moveUp(Character& character);
14     virtual InputInfo moveDown(Character& character);
15     virtual InputInfo moveLeft(Character& character);
16     virtual InputInfo moveRight(Character& character);
17     virtual InputInfo stopMove(Character& character);
18
19     virtual InputInfo selectItem(Character& character, int item);
20     virtual InputInfo dropItem(Character& character, int item);
21     virtual InputInfo unequipItem(Character& character, int item);
22     virtual InputInfo buyItem(Character& character, int item);
23     virtual InputInfo sellItem(Character& character, int item);
24
25     virtual InputInfo deposit(Character& character,int item, bool isItem);
26     virtual InputInfo retire(Character& character,int item, bool isItem);
27
28     virtual InputInfo meditate(Character& character);
29     virtual InputInfo resurrect(Character& character);
30     virtual InputInfo cure(Character& character);
31     virtual InputInfo takeItem(Character& character);
32     virtual InputInfo selectTarget(Character& character, Point position);
33 };
34
35 #endif
```

jul 21, 20 15:20 **InteractState.cpp** Page 1/3

```cpp
#include "InteractState.h"

InteractState::InteractState() :
    CharacterState(CharacterStateID::Interact), beforeInput(InputID::nothing){}

InteractState::~InteractState() {}

InputInfo InteractState::moveUp(Character& character){
    InputInfo info;
    info.input = InputID::up;
    if (info.input == beforeInput) {
        info.input = InputID::nothing;
    }
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::moveDown(Character& character) {
    InputInfo info;
    info.input = InputID::down;
    if (info.input == beforeInput) {
        info.input = InputID::nothing;
    }
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::moveLeft(Character& character) {
    InputInfo info;
    info.input = InputID::left;
    if (info.input == beforeInput) {
        info.input = InputID::nothing;
    }
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::moveRight(Character& character) {
    InputInfo info;
    info.input = InputID::right;
    if (info.input == beforeInput) {
        info.input = InputID::nothing;
    }
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::stopMove(Character& character) {
    InputInfo info;
    info.input = InputID::stopMove;
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::selectItem(Character& character, int item) {
    InputInfo info;
```

jul 21, 20 15:20 **InteractState.cpp** Page 2/3

```cpp
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::selectTarget(Character& character, Point position) {
    InputInfo info;
    info.position = position;
    info.input =InputID::selectTarget;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::meditate(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::resurrect(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::resurrect;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::cure(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::cure;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::takeItem(Character& character) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::dropItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    beforeInput = info.input;
    return info;
}

InputInfo InteractState::buyItem(Character& character,int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::buy;
    info.aditional = item;
```

jul 21, 20 15:20 **InteractState.cpp** Page 3/3

```
133     beforeInput = info.input;
134     return info;
135 }
136
137 InputInfo InteractState::sellItem(Character& character,int item) {
138     InputInfo info;
139     Point aux(0.0,0.0);
140     info.position = aux;
141     info.input =InputID::sell;
142     info.aditional = item;
143     beforeInput = info.input;
144     return info;
145 }
146
147 InputInfo InteractState::deposit(Character& character,int item, bool isItem) {
148     InputInfo info;
149     Point aux(0.0,0.0);
150     info.position = aux;
151     if (isItem) {
152         info.input =InputID::depositItem;
153     } else {
154         info.input =InputID::depositGold;
155     }
156     info.aditional = item;
157     beforeInput = info.input;
158     return info;
159 }
160
161 InputInfo InteractState::retire(Character& character,int item, bool isItem) {
162     InputInfo info;
163     Point aux(0.0,0.0);
164     info.position = aux;
165     if (isItem) {
166         info.input =InputID::retireItem;
167     } else {
168         info.input =InputID::retireGold;
169     }
170     info.aditional = item;
171     beforeInput = info.input;
172     return info;
173 }
174
175 InputInfo InteractState::unequipItem(Character& character,int item) {
176     InputInfo info;
177     Point aux(0.0,0.0);
178     info.position = aux;
179     info.input = InputID::nothing;
180     beforeInput = info.input;
181     return info;
182 }
183
```

jul 21, 20 15:20 **CharacterState.h** Page 1/1

```
1  #ifndef CHARACTERSTATE_H
2  #define CHARACTERSTATE_H
3
4  #include "../../common/Identificators.h"
5  #include "../Character.h"
6
7  //Esta clase es la utilizada por los personajes del juego para retratar su
8  //estado. Las implementaciones de esta clase abstracta
9  //permitirÃ¡n o no el efecto de ciertos comandos.
10
11 class CharacterState {
12 private:
13     CharacterStateID id;
14 public:
15     explicit CharacterState(CharacterStateID state) : id(state) {}
16     virtual ~CharacterState() = default;
17     virtual CharacterStateID& getState() {
18         return this→id;
19     }
20
21     virtual InputInfo moveUp(Character& character) = 0;
22     virtual InputInfo moveDown(Character& character) = 0;
23     virtual InputInfo moveLeft(Character& character) = 0;
24     virtual InputInfo moveRight(Character& character) = 0;
25     virtual InputInfo stopMove(Character& character) = 0;
26
27     virtual InputInfo selectItem(Character& character, int item) = 0;
28     virtual InputInfo unequipItem(Character& character, int item) = 0;
29     virtual InputInfo dropItem(Character& character, int item) = 0;
30     virtual InputInfo takeItem(Character& character) = 0;
31     virtual InputInfo resurrect(Character& character) = 0;
32     virtual InputInfo meditate(Character& character) = 0;
33     virtual InputInfo cure(Character& character) = 0;
34     virtual InputInfo buyItem(Character& character, int item) = 0;
35     virtual InputInfo sellItem(Character& character, int item) = 0;
36     virtual InputInfo deposit(Character& character,int item, bool isItem) = 0;
37     virtual InputInfo retire(Character& character,int item, bool isItem) = 0;
38     virtual InputInfo selectTarget(Character& character, Point position) = 0;
39
40 };
41
42 #endif
```

jul 21, 20 15:20 **AttackState.h** Page 1/1

```
#ifndef ATTACKSTATE_H
#define ATTACKSTATE_H

#include "CharacterState.h"

class AttackState: public CharacterState {
private:
    InputID beforeInput;
public:
    AttackState();
    ~AttackState();

    virtual InputInfo moveUp(Character& character);
    virtual InputInfo moveDown(Character& character);
    virtual InputInfo moveLeft(Character& character);
    virtual InputInfo moveRight(Character& character);
    virtual InputInfo stopMove(Character& character);

    virtual InputInfo selectItem(Character& character, int item);
    virtual InputInfo dropItem(Character& character, int item);
    virtual InputInfo unequipItem(Character& character, int item);
    virtual InputInfo buyItem(Character& character, int item);
    virtual InputInfo sellItem(Character& character, int item);

    virtual InputInfo deposit(Character& character,int item, bool isItem);
    virtual InputInfo retire(Character& character,int item, bool isItem);

    virtual InputInfo meditate(Character& character);
    virtual InputInfo resurrect(Character& character);
    virtual InputInfo cure(Character& character);
    virtual InputInfo takeItem(Character& character);
    virtual InputInfo selectTarget(Character& character, Point position);
};

#endif
```

jul 21, 20 15:20 **AttackState.cpp** Page 1/3

```
#include "AttackState.h"

AttackState::AttackState() :
    CharacterState(CharacterStateID::Attack) , beforeInput(InputID::nothing) {}

AttackState::~AttackState() = default;

InputInfo AttackState::moveUp(Character& character){
    InputInfo info;
    info.input = InputID::up;
    if (character.getDirection() == Direction::up)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo AttackState::moveDown(Character& character) {
    InputInfo info;
    info.input = InputID::down;
    if (character.getDirection() == Direction::down)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo AttackState::moveLeft(Character& character) {
    InputInfo info;
    info.input = InputID::left;
    if (character.getDirection() == Direction::left)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo AttackState::moveRight(Character& character) {
    InputInfo info;
    info.input = InputID::right;
    if (character.getDirection() == Direction::right)
        info.input = InputID::nothing;
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo AttackState::stopMove(Character& character) {
    InputInfo info;
    info.input = InputID::stopMove;
    Point aux(0.0,0.0);
    info.position = aux;
    beforeInput = info.input;
    return info;
}

InputInfo AttackState::selectItem(Character& character, int item) {
    InputInfo info;
    Point aux(0.0,0.0);
    info.position = aux;
    info.input =InputID::nothing;
    beforeInput = info.input;
```

```
67      return info;
68  }
69
70  InputInfo AttackState::selectTarget(Character& character, Point position) {
71      InputInfo info;
72      info.position = position;
73      info.input = InputID::selectTarget;
74      if (beforeInput == info.input)
75          info.input = InputID::nothing;
76      beforeInput = info.input;
77      return info;
78  }
79
80  InputInfo AttackState::meditate(Character& character) {
81      InputInfo info;
82      Point aux(0.0,0.0);
83      info.position = aux;
84      info.input =InputID::nothing;
85      beforeInput = info.input;
86      return info;
87  }
88
89  InputInfo AttackState::resurrect(Character& character) {
90      InputInfo info;
91      Point aux(0.0,0.0);
92      info.position = aux;
93      info.input =InputID::nothing;
94      beforeInput = info.input;
95      return info;
96  }
97
98  InputInfo AttackState::cure(Character& character) {
99      InputInfo info;
100     Point aux(0.0,0.0);
101     info.position = aux;
102     info.input =InputID::nothing;
103     beforeInput = info.input;
104     return info;
105 }
106
107 InputInfo AttackState::takeItem(Character& character) {
108     InputInfo info;
109     Point aux(0.0,0.0);
110     info.position = aux;
111     info.input =InputID::nothing;
112     beforeInput = info.input;
113     return info;
114 }
115
116 InputInfo AttackState::dropItem(Character& character, int item) {
117     InputInfo info;
118     Point aux(0.0,0.0);
119     info.position = aux;
120     info.input =InputID::nothing;
121     beforeInput = info.input;
122     return info;
123 }
124
125 InputInfo AttackState::buyItem(Character& character,int item) {
126     InputInfo info;
127     Point aux(0.0,0.0);
128     info.position = aux;
129     info.input =InputID::nothing;
130     beforeInput = info.input;
131     return info;
132 }
```



```
133
134 InputInfo AttackState::sellItem(Character& character,int item) {
135     InputInfo info;
136     Point aux(0.0,0.0);
137     info.position = aux;
138     info.input =InputID::nothing;
139     beforeInput = info.input;
140     return info;
141 }
142
143 InputInfo AttackState::retire(Character& character,int item, bool isItem) {
144     InputInfo info;
145     Point aux(0.0,0.0);
146     info.position = aux;
147     info.input = InputID::nothing;
148     beforeInput = info.input;
149     return info;
150 }
151
152 InputInfo AttackState::deposit(Character& character,int item, bool isItem) {
153     InputInfo info;
154     Point aux(0.0,0.0);
155     info.position = aux;
156     info.input = InputID::nothing;
157     beforeInput = info.input;
158     return info;
159 }
160
161 InputInfo AttackState::unequipItem(Character& character,int item) {
162     InputInfo info;
163     Point aux(0.0,0.0);
164     info.position = aux;
165     info.input = InputID::nothing;
166     beforeInput = info.input;
167     return info;
168 }
```

jul 21, 20 15:20 **Camera.h** Page 1/1

```h
#ifndef CAMERA_H
#define CAMERA_H

#include <SDL2/SDL.h>
#include "Window.h"
#include "../common/Point.h"

//Clase destinada a mantener centrada la camara del juego sobre
//el jugador que es seteado como target.

class Camera {
private:
    Window& window;
    float width, height; //Limites del mapa
    float scale;
    Point* playerTarget = nullptr; //Jugador en el cual se debe centrar la camara.
    Point positionScreen; //
    SDL_Rect cam; //Dimensiones de la cÃ¡mara.

    //Ajusta la posiciÃ³n destiny para que nunca se pase de los lÃmites del mapa.
    void limits(Point* destiny);
public:
    Camera(Window& window, float widthMap, float heightMap);

    SDL_Rect getCamera() const;

    float getCameraWidth() const;

    float getCameraHeight() const;

    Point getCameraPosition() const;

    float getScale() const;

    void setPlayer(Point* player);

    void render(Point destiny);

    //Devuelve verdadero en caso de que el click haya sido realizado
    //dentro del Viewport de la camara.
    bool clickInMap(Point coordinates) const;

    //Devuelve un Point de acuerdo a las coordenadas globales,
    //sacandole el relativo de la camara.
    Point calculateGlobalPosition(Point coordinates) const;

    //Devuelve la distancia que hay entre el Point coordinates y
    //el target que tiene ajustado la cÃ¡mara.
    int distanceFromTarget(Point coordinates) const;

    ~Camera();
};

#endif
```

jul 21, 20 15:20 **Camera.cpp** Page 1/2

```cpp
#include "Camera.h"
#include <algorithm>
#include <SDL2/SDL.h>

#define WIDTHSEGMENT 8
#define TOPBARHEIGHT 60

Camera::Camera(Window& window, float widthMap, float heightMap) : window(window),
    width(widthMap), height(heightMap), scale(1.0f) {
    this→positionScreen = Point(0.0,float(this→window.getHeight())/2.0);
    this→cam = {(this→window.getWidth()/WIDTHSEGMENT)*2,TOPBARHEIGHT,(this→window.getWidth()/WIDTHSEGMENT)*6, this→window.getHeight()−TOPBARHEIGHT};
}

float Camera::getCameraWidth() const {
    return float(this→window.getWidth());
}

float Camera::getCameraHeight() const {
    return float(this→window.getHeight());
}

SDL_Rect Camera::getCamera() const {
    return this→cam;
}

float Camera::getScale() const{
    return this→scale;
}

Point Camera::getCameraPosition() const {
    return this→positionScreen;
}

void Camera::setPlayer(Point* player){
    this→playerTarget = player;
}

void Camera::limits(Point* destiny) {
    if (this→playerTarget ≠ nullptr) {
        float limitWidth = ((this→window.getWidth()/WIDTHSEGMENT)*6) / 2.0f;
        float limitHeight = (this→window.getHeight()−TOPBARHEIGHT) / 2.0f;

        destiny→y = std::max(destiny→y,limitHeight);
        destiny→y = std::min(destiny→y, this→height−limitHeight);
        destiny→x= std::min(destiny→x, this→width−limitWidth);
        destiny→x= std::max(destiny→x, limitWidth);
    }

}

Point Camera::calculateGlobalPosition(Point coordinates) const {
    float x = coordinates.x +(positionScreen.x−cam.x);
    float y = coordinates.y + (positionScreen.y−cam.y);
    return Point(x,y);
}

bool Camera::clickInMap(Point coordinates) const {
    bool inMap = false;
    if (coordinates.x > cam.x ∧ coordinates.x < cam.x+cam.w ∧
        coordinates.y > cam.y ∧ coordinates.y < cam.y+cam.h)
        inMap = true;
    return inMap;
}
```

jul 21, 20 15:20 **Camera.cpp** Page 2/2

```
65  void Camera::render(Point destiny) {
66      limits(&destiny);
67      this→cam = {(this→window.getWidth()/WIDTHSEGMENT)*2,TOPBARHEIGHT,(this→win
    dow.getWidth()/WIDTHSEGMENT)*6,
68                  this→window.getHeight()−TOPBARHEIGHT};
69      this→positionScreen.x = destiny.x − (((this→window.getWidth()/WIDTHSEGMENT
    )*6) / 2.0f);
70      this→positionScreen.y = destiny.y − ((this→window.getHeight() − TOPBARHEIG
    HT) / 2.0f);
71      SDL_Rect display = {(this→window.getWidth()/WIDTHSEGMENT) * 2,TOPBARHEIGHT,
72              (this→window.getWidth()/WIDTHSEGMENT) * 6,this→window.getHeight()
    − TOPBARHEIGHT};
73      SDL_RenderSetViewport(&(this→window.getRenderer()), &display);
74  }
75
76  int Camera::distanceFromTarget(Point coordinates) const {
77      return this→playerTarget→distance(coordinates);
78  }
79
80  Camera::~Camera()= default;
```

jul 21, 20 15:20 **Table of Content** Page 1/5

Table of Contents