Taller de Programación I (75.42 / 95.08)

Jazz Jackrabbit 2

Trabajo Práctico Grupal - Documentación Técnica

Curso Veiga - Grupo 7

Integrantes

Avalos, Victoria	108434
Castro Martinez, José Ignacio	106957
Diem, Walter Gabriel	105618

Índice

Servidor

El servidor corresponde a una pieza fundamental de la aplicación el cual habilita a las distintas computadoras a comunicarse mediante sockets TCP. Para la construcción del servidor se realizó un diseño considerando las necesidades en la agilidad para la comunicación entre el cliente y el servidor. El servidor está compuesto principalmente de dos piezas:

- Un monitor de partidas
- Un hilo aceptador

El monitor de partidas está encargado de funcionar como wrapper y administrador del recurso compartido más importante: los juegos, estos están contenidos en forma de una clase llamada game Wrapper que almacena todos los datos vitales de la partida y necesarios para poder identificarla, así como también del hilo del game loop y las colas necesarias para realizar la comunicación entre los distintos hilos.

El hilo aceptador por otro lado es el manejador de los clientes que se conectan al servidor, contiene a los hilos Sender y Receiver de la partida y gestiona las comunicaciones y los estados en que se encuentran los mismos.

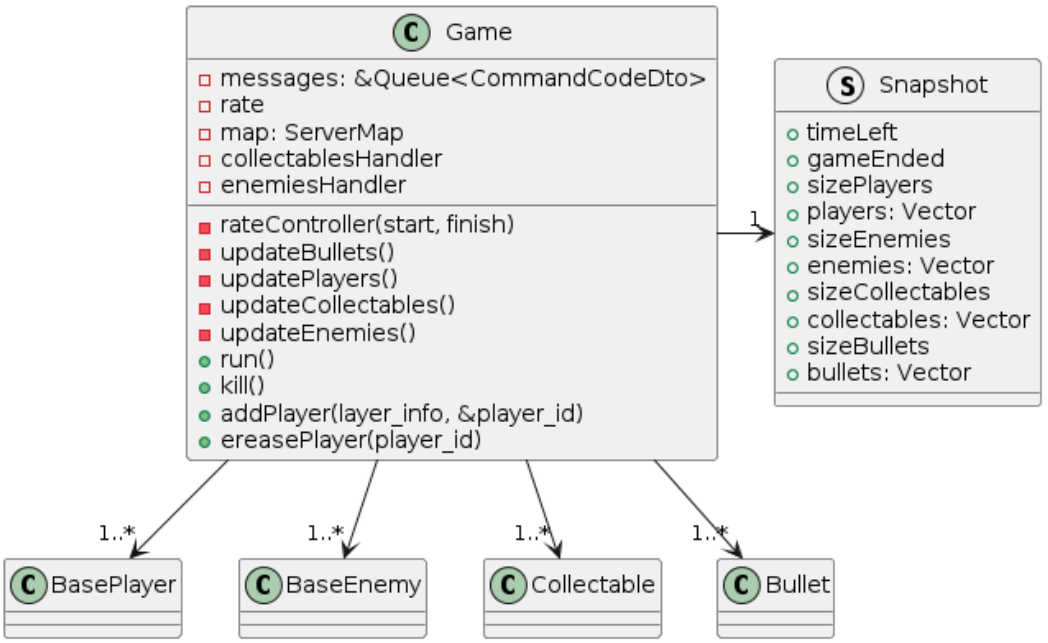
Los hilos de comunicación sender y receiver son hilos que utilizan colas bloqueantes para recibir y enviar mensaje entre cliente y servidor, ambos corren independientes del resto de la lógica y utilizan el protocolo del servidor para poder recibir la información mediante socket.

Respecto a la integración con el engine, se cuenta con la clase Game que posee un mapa con los distintos jugadores. Esta clase es la que lleva a cabo el loop principal del juego con un rate constante. Por cada iteración se procesan un máximo configurable de instrucciones, establecido a priori como 100. En base a la instrucción, se invoca al método de BasePlayer correspondiente, excepto para los cheats que realizan otras funciones en el juego no relacionadas al jugador.

El Game posee el mapa, un vector de balas, uno de enemigos y uno de coleccionables. Cuenta con un método de actualización para cada vector, en los cuales se verifican las intersecciones: si una bala intersecta a un jugador o a un enemigo para hacerle daño, si un jugador intersecta con un ítem para recolectarlo, si un jugador está realizando una acción especial e intersecta con otro jugador o con un enemigo para realizar daño, y si un jugador está en el rango de un enemigo para hacerle daño. Luego, se invoca al método correspondiente.

Se posee un único struct Snapshot que se modifica con cada acción, y la misma instancia se envía al cliente constantemente. El mismo posee vectores de otros structs, un struct por cada tipo de entidad: jugador, enemigo, coleccionable y bala. Todas las entidades conocen al snapshot y modifican los flags correspondientes cuando sucede una acción. Por ejemplo, cuando un jugador se mueve, modifica su posición en la snapshot.

A continuación se muestra un diagrama de clases con los métodos y atributos más importantes. En el apartado de Engine, se entra más en detalle respecto de las entidades y su funcionamiento.



Cliente

En éste apartado se hablará únicamente del back-end del cliente, el cual posee dos componentes principales: el Lobby y el cliente.

El Lobby posee una estructura secuencial sin la implementación de hilos. Consiste de una clase Lobby que posee como atributo a la clase LobbyProtocol. La comunicación a través del socket se delega al protocolo.

El mismo realiza las siguientes acciones:

1. Realizar la conexión con el servidor mediante un socket.
2. Recibir la información de los juegos disponibles.
3. Refrescar (avisar al servidor y volver a recibir) dicha información si el usuario presionó el botón de refresh.
4. Enviar el juego seleccionado. Si el jugador creó una partida en lugar de unirse a una, la información del juego seleccionado se envía de la misma forma, y es el servidor el que se encarga de procesarlo como una partida nueva.
5. Recibir el id que el servidor le asignó al jugador.

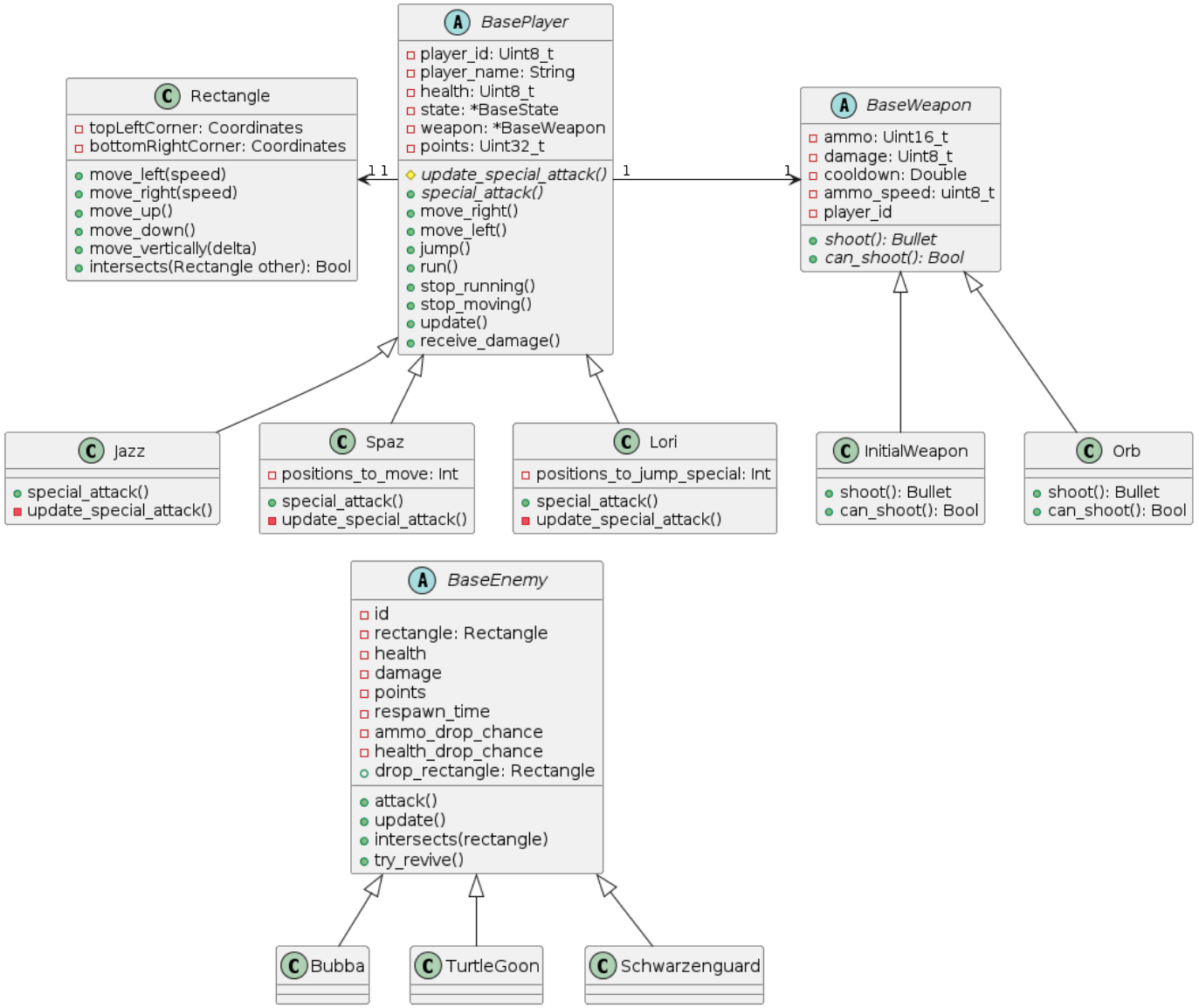
El Lobby libera los recursos del socket si el usuario cierra el juego antes de que muera el Lobby.

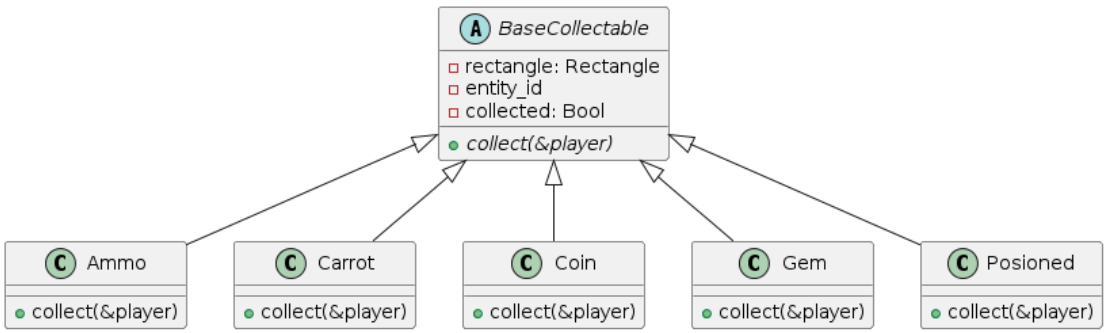
Luego, el ownership del socket utilizado para la conexión con el servidor es transferido al Client y el Lobby no se utiliza más.

El Client cuenta con una arquitectura basada en hilos: posee un hilo receiver y uno sender, de forma similar al servidor pero sin un aceptador. Se maneja con colas de snapshots, para recibir mensajes, y otra de acciones para enviarlos. Posee un protocolo que maneja el endiannes del snapshot, y dicho protocolo se encarga de las comunicaciones vía socket.

Engine

El engine del juego cuenta con los siguientes elementos: players, weapons, states, enemies, collectables y bullets. Para todos ellos, excepto bullets, se utiliza polimorfismo para manejar los diferentes comportamientos y cuentan con una clase base abstracta. Se utilizan punteros inteligentes a las clases base. Un jugador tiene un arma y un estado, que cambian dinámicamente en base a lo que suceda en el juego. El arma crea una bullet que se agrega al vector de balas en el Game, a la cual le inyecta el daño y velocidad correspondientes, junto con el id del jugador que la disparó.
A continuación se muestran diagramas de clase con algunos de los métodos y atributos más importantes:





En los enemigos, las clases hijas setean los atributos correspondientes. De forma similar, con los estados hay una clase padre abstracta y las clases hijas determinan si se pueden realizar ciertas acciones, como disparar o moverse.

Se utiliza un contador global con el patrón singleton para crear ids únicos para todas las entidades.

Respecto del manejo de colisiones, se utiliza el sistema conocido como AABB. Cada entidad tiene un rectángulo, el cual posee una esquina superior izquierda y una esquina inferior derecha. Para identificar una intersección, la clase Rectangle posee un método intersects que se compara a sí mismo con otro rectángulo en base a dichas esquinas para determinar si están colisionando o no.

El jugador posee un método de update que, entre muchas otras cosas, intenta mover hacia abajo su rectángulo, y se fija si hay una intersección con el mapa. El mapa del juego es un vector de rectángulos que representan el piso, las paredes, el techo y las plataformas. De ésta forma se simula una “gravedad”.

Interfaz gráfica

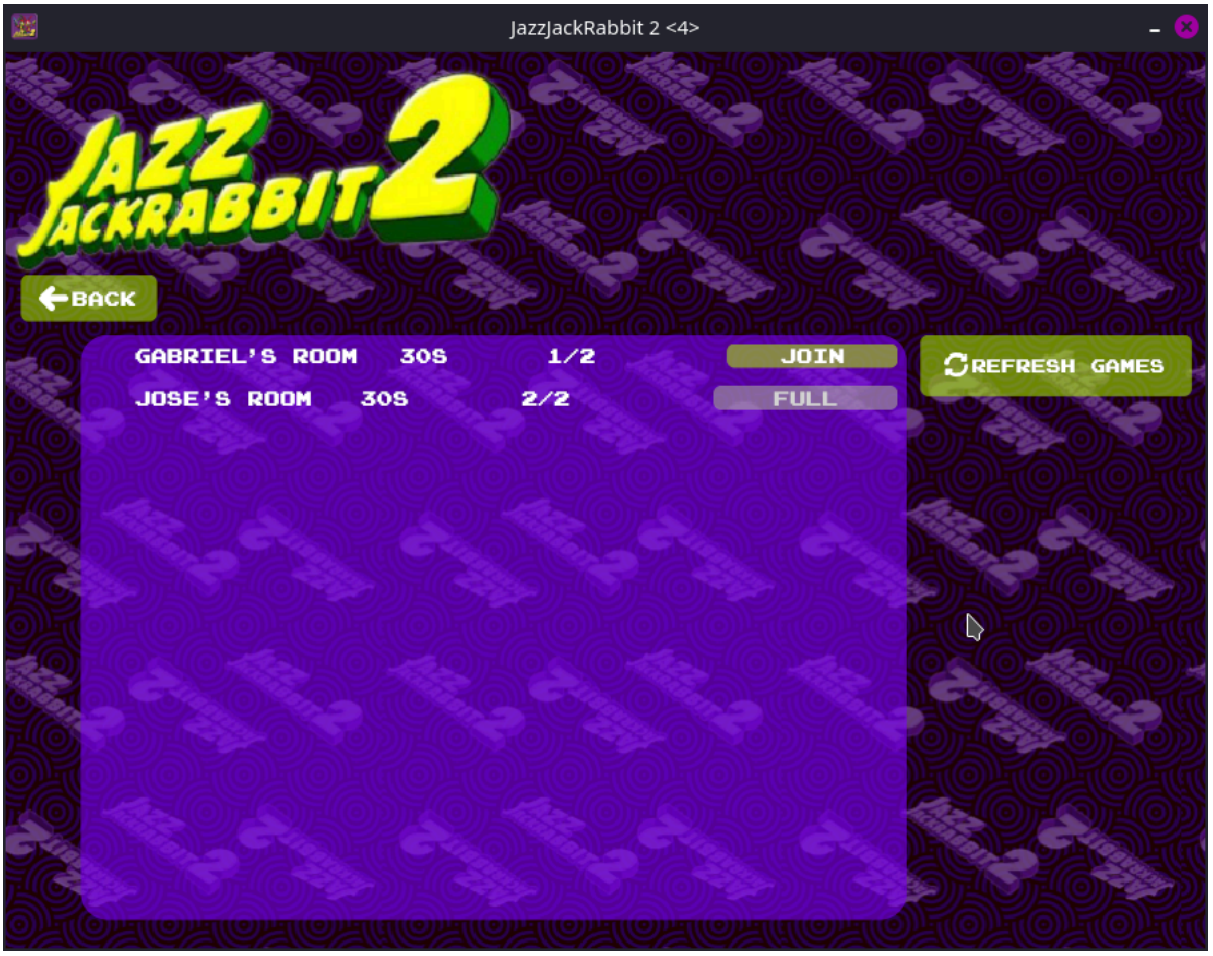
El juego consta de 2 ventanas gráficas principales, el Lobby y el Game.

Lobby

Es la primera ventana que aparece al iniciar el cliente. Construida íntegramente en Qt5 mediante la fusión del XML generado con Qt Creator y código custom para generar elementos on-the-fly de manera dinámica.

La ventana contiene un StackedWidget, donde cada “pantalla” distinta en un Widget del StackedWidget que se va cambiando.

Se encuentra en el código como la clase StartupScreen, esta se conecta con la clase Lobby, que posee el LobbyProtocol encargado de hacer las comunicaciones con el servidor mediante los comandos.



El Widget de la sala de juegos se genera dinámicamente en base a la información recibida por el server al inicio de la conexión o al presionar el botón de Refresh Games.

Cada input, ya sea de determinados botones como el botón de Join, como los LineEdits primero son validados para ver si se puede proceder correctamente. Como ejemplo, en el caso del LineEdit de username de valida su longitud.
En el caso particular del botón de Join se hace un refresh de los juegos apenas de lo presiona para ver si el juego sigue con espacio disponible, sino se le avisa al usuario que está lleno.

En la ventana de espera, se le delega la responsabilidad de cerrar la StartupScreen a un thread específico que se bloquea hasta recibir la primera snapshot del server, en términos del protocolo esto significa que todos los jugadores se unieron y comienza la partida. De esta manera no se bloquea el thread de la UI de Qt.

Al seleccionar un personaje se reproduce el gif del mismo, realizando un movimiento.

Cada botón del Lobby tiene un sonido de click asociado. También está implementado la música de fondo que es controlada por el archivo de configuración YAML.

Game

Una vez se cierra el Lobby porque se recibió la primera snapshot, se precarga el GraphicEngine, el AudioEngine, el mapa y el Hud para comenzar el renderizado por el Renderer.

El GraphicEngine mediante el TextureLoader precarga los sprites en memoria, de esta manera cada clase que necesite renderizar un sprite utilizará una referencia en lugar de cargar de nuevo un sprite desde el archivo.

Un Sprite consta de la textura de SDL almacenada en memoria más los datos de cómo usar el Sprite que se cargaron de un archivo yaml de metadata. Esa metadata cuenta con la cantidad de frames y la posición de los mismos en el spritesheet.

El AudioEngine funciona de manera similar al GraphicEngine pero con los audios, haciendo diferencia entre SoundEffects (sonidos cortos) y Music (piezas de música largas). Cuando

se solicita al engine que reproduzca un sonido, lanzará un thread para reproducirlo, y lo joineará una vez terminada la reproducción.

El Map (mapa) realiza la carga de las coordenadas desde un archivo yaml de coordenadas, donde se diferencia entre bloques de tierra (full dirt), bloques con pasto (top grass) y pendientes (slopes). Esto se usa para luego renderizar los sprites correspondientes al tipo de bloque. La cámara se mueve de acuerdo a la posición de jugador (clase Player) y a su ubicación relativa a la esquina superior izquierda.

El Renderer maneja el constant rate loop utilizado para el renderizado, renderizando cada uno de los Renderables existentes y creando nuevos si se debiera. Se detecta que un nuevo Renderable debe ser creado cuando no es un renderable ya existente. Y se detecta que uno debe ser eliminado cuando el mismo renderable responde con true al método shouldDelete.
Bajo la interfaz Renderable se encuentran: Jazz, Spaz y Lori (que a su vez responden a la interfaz Playable Character), Bubba, Turtle Goon, Schwarzenguard, Carrot, Coin, Diamond, Ammo y Bullets.