Documentación Técnica

Taller I Veiga - 2do Cuatrimestre 2017
Ayudante: Martín Di Paola

Requerimientos de software

OS: GNU/Linux
Dependencias:

- SDL_2.0 (sudo apt-get install libsdl2-dev)
- SDL_Image (sudo apt-get install libsdl2-image-dev)
- SDL_ttf (sudo apt-get install libsdl2-ttf-dev)
- GTK3 (sudo apt-get install libgtk-3-dev)
- gtkmm (sudo apt-get install libgtkmm-3.0-dev)
- yaml-cpp (sudo apt-get install libyaml-cpp-dev)
- Boost C++ -dependencia de yaml-(sudo apt-get install libboost-dev)
- CMake versión 3.5 (sudo apt-get install cmake)

Depuración:

- GDB y Valgrind (sudo apt-get install libc6-dbg gdb valgrind)

Descripción general

El proyecto consta de tres partes claramente diferenciadas. A grandes rasgos, tenemos el **Servidor**, que cuenta con la lógica del juego y maneja las conexiones de los clientes. Por otro lado está el **Cliente** que recibe datos del servidor y los grafica por pantalla y también envía comandos al servidor. Y por último, está el **Editor** que sirve para crear los distintos niveles.

Cliente

El cliente está hecho en GTKmm para el lobby y en SDL para el juego.

Para el juego, la clase primordial es **Renderer** que se encarga de dibujar todo lo que se ve en la pantalla. Para ello cuenta con:

-**Métodos de cambios de coordenadas:** cambia de isométricas a cartesianas y viceversa tanto para graficar como para manejar un click del usuario

- int cartesianToIsometricX(int x, int y);
- int cartesianToIsometricY(int x, int y);
- int pixelToCartesianX(int x, int y);
- int pixelToCartesianY(int x, int y);

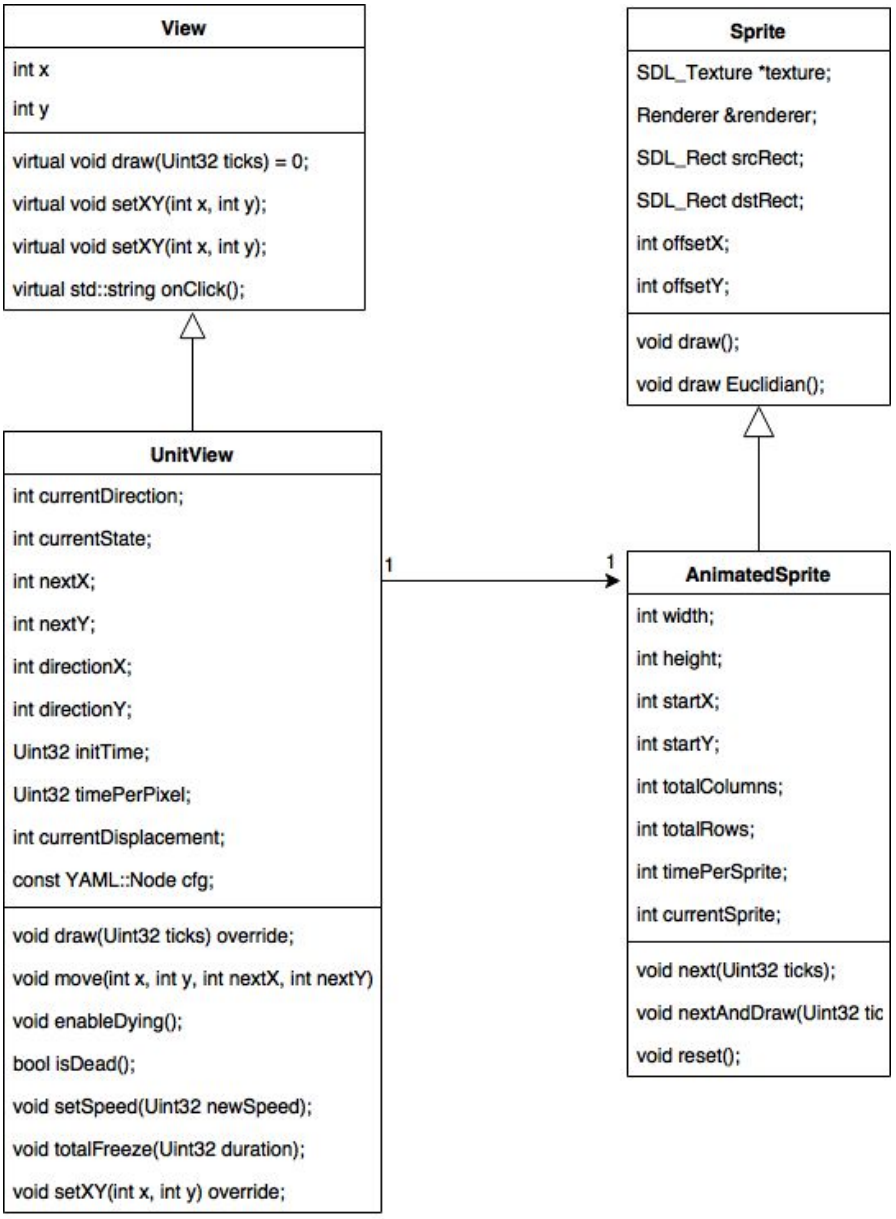
-**Métodos para copiar una textura por pantalla:** tanto con proyección isométrica como cartesiana. Es interesante notar que para poder implementar el zoom in y el zoom out estos métodos fueron cruciales.

- void copy(SDL_Texture *texture, const SDL_Rect *src,
  SDL_Rect *dst, int offsetX, int offsetY);
- void copyEuclidian(SDL_Texture *texture, SDL_Rect *src, SDL_Rect *dst)

-**Métodos para actualizar y administrar la cámara:** al ser la encargada de mostrar las texturas por pantalla, la clase Renderer debe encargarse de no dibujar todo el mapa, sino sólo lo que se ve de acuerdo a las dimensiones de la pantalla del usuario.

- void updateCamera(int x, int y);
- bool isOnCamera(int x, int y);
- void updateCameraFinger(int x, int y);

Diagrama de las clases más importantes de la vista:



Arquitectura Cliente Servidor

Servidor:

El servidor consta de un thread dedicado para aceptar nuevas conexiones. Acepta las conexiones y inicializa las estructuras asociadas para luego lanzar un thread que escuchara los comandos del cliente. Por otro lado, el servidor lanza un thread encargado de notificara todos los jugadores que NO estén en una partida de los cambios y updates respecto a los lobbies y sus jugadores. Una vez que el servidor tiene un lobby con todos los jugadores listos para jugar, lanza un thread para el modelo del juego y un thread de notificación. Este thread de notificación únicamente levanta las notificaciones asociadas a su juego y por ende únicamente notifica a aquellos jugadores que estén dentro del mismo.

El thread recibidor de cada cliente recibe la instrucción a ejecutar, y actúa acordemente. En el caso de una instrucción para un juego en curso, este se traduce en un comando encolado en una cola no bloqueante (thread safe) dentro del juego. En cada ciclo del juego, levanta todas los comandos y los ejecuta, en caso de haber modificado el modelo interno del juego, el thread del modelo del juego encola un comando de notificación en la cola de notificaciones asociada. Esta cola es bloqueante, porque solo notificamos cuando hay algún cambio del modelo para notificar.

Cliente:

El cliente podría dividirse entre las partes que ejecutan el lobby (GTKmm) y el juego (SDL). Estas dos partes comparten 3 piezas clave, un despachador de comandos con una cola bloqueante que envía los comandos al servidor en un thread aparte, un recibidor de notificaciones que recibe notificaciones del servidor en un thread aparte, y por último el lobby manager del lado del cliente.

Los tanto GTK como SDL encolan los comandos (ya sean de juego, o de lobby) en la cola bloqueante del despachador. Por otro lado el thread recibidor altera de manera thread-safe el estado del lobby manager y el modelo de la vista.

Editor

El editor es una entidad aparte del cliente. No necesita conectarse al servidor, por lo que su arquitectura resulta mucho más sencilla. Simplemente se reutiliza la parte gráfica del cliente y, con ayuda de otros modelos auxiliares se pudo llegar al resultado.

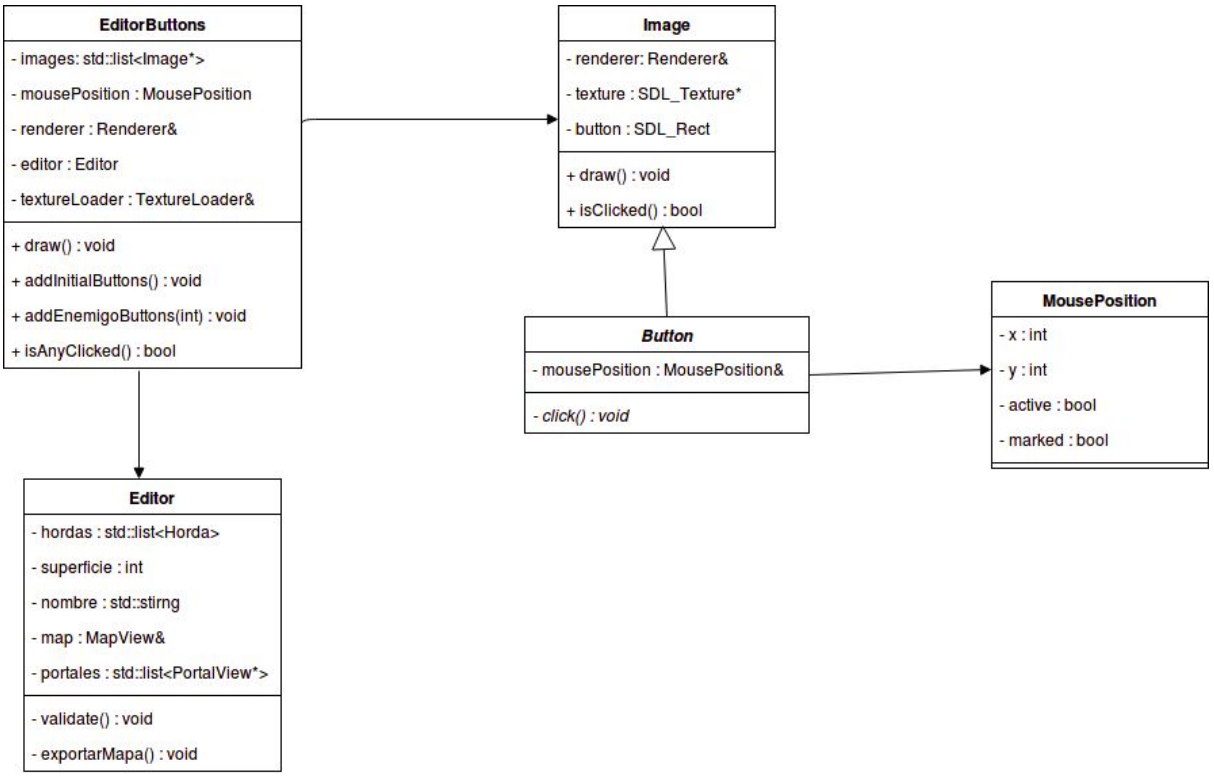
Para empezar, modelamos los botones mediante una clase abstracta Buttons, la cual sabe en qué posición se encuentra, su imagen, etc. En cada ciclo de dibujado el botón debe ver si está clickeado, y si lo está, ejecutar su función abstracta click(). De esta forma se crearon los distintos botones del juego, sobreescribiendo el método click y, en algunos casos, agregando comportamiento en el método draw().

El ciclo principal es prácticamente idéntico al del cliente, cambiando solamente los botones utilizados y los distintos eventos posibles. Para facilitar el manejo de los botones (ya que la cantidad puede variar durante el ejecución, debido a la cantidad de hordas) encapsulamos una lista, formando la clase EditorButtons. Esta clase es la que se encarga de agregar/eliminar botones, de dibujarlos, etc. De todas formas, a la lista de botones no se le puede agregar cualquier botón: solamente se pueden agregar utilizando los métodos de la clase, como puede ser addEnemigosButtons, que agrega todos los botones de una horda.

Otro de los desafíos fue saber en qué momento se debe ejecutar el comportamiento de los botones. Para esto se creó un modelo que guarda, cada vez que se hace click, la posición del mouse. De esta forma, los botones saben si fueron clickeados en cada ciclo de dibujado.

Por último, debíamos guardar los datos de nuestro mapa, por lo que creamos la entidad Editor, a la cual se le delega la validación y exportación del mismo.

A continuación se muestra un diagrama de clases con las entidades más importantes que constituyen el editor:


EditorButtons
- images: std::list<Image*>
- mousePosition : MousePosition
- renderer : Renderer&
- editor : Editor
- textureLoader : TextureLoader&
+ draw() : void
+ addInitialButtons() : void
+ addEnemigoButtons(int) : void
+ isAnyClicked() : bool
Image
- renderer: Renderer&
- texture : SDL_Texture*
- button : SDL_Rect
+ draw() : void
+ isClicked() : bool
Button
- mousePosition : MousePosition&
- click() : void
MousePosition
- x : int
- y : int
- active : bool
- marked : bool
Editor
- hordas : std::list<Horda>
- superficie : int
- nombre : std::stirng
- map : MapView&
- portales : std::list<PortalView*>
- validate() : void
- exportarMapa() : void