Argentum Online
Manual de Proyecto

Taller de programacion I [75.42]
Primer cuatrimestre de 2020

Taiel Colavecchia - 102510 - tcolavecchia@fi.uba.ar
Franco Daniel Schischlo - 100615 - fschischlo@fi.uba.ar
Nicolás Aguerre - 102145 - naguerre@fi.uba.ar

Índice

1. División de tareas

En un primer momento, las tareas fueron divididas entre los tres integrantes mencionados en la carátula de la siguiente forma (en un aspecto mas general, que será detallado mas adelante en la presente sección).

- **Nicolás Aguerre**: Construcción general del cliente y manejo de gráficos.
- **Franco Daniel Schischlo**: Contenido y logica de juego en el servidor. Persistencia.
- **Taiel Colavecchia**: Arquitectura de comunicación y del servidor. Conteniedo y lógica del servidor.

A continuación, se indica un desgloce mas detallado de las tareas realizadas por cada integrante.

1.1. Franco Daniel Schischlo

- Persistencia de los personajes con toda su información relevante.
- Diseño y creación de las distintas databases (json's) de los items, clases, razas, etc, junto a las clases que se encargan de manejarlas.
- Diseño y planteo en parte de la clase Entity, Item, y sus distintos subtipos.
- Creación de un contenedor de items (inventario).
- Generador de eventos random para el momento del drop.
- Diseño de un Entity Component System que se usa del lado del cliente.

1.2. Taiel Colavecchia

- Diseño y construcción de la arquitectura de comunicación cliente-servidor.
  - Diseñar y construir una estructura que permitiera enviar o recibir datos de forma concurrente, esto implicó crear una clase para el manejo de un *Socket* a través del uso de dos Threads. Este puede ser utilizado tanto para el programa cliente y para atender a cada cliente del lado del servidor.
  - Construcción de las clases que permiten la comunicación a través de un Protocolo flexible entre los clientes y el servidor utilizando una biblioteca para el manejo de objetos de tipo JSON.
- Construcción de la arquitectura concurrente del servidor (el diseño fue planteado y diseñado en conjunto con los otros integrantes). Esto abarca:
  - Aceptación de nuevos clientes. Simplemente a través de un Thread dedicado a aceptar clientes y permitir que envíen un mensaje inicial.
  - Manipulación de los Sockets que atienden a cada cliente: se deben recibir los eventos enviados por cada uno y enviarle las actualizaciones y mensajes correspondientes a cada uno.
  - Actualización lógica del juego, de forma aislada a los clientes y los eventos que pudiesen ser enviados por los mismos.
  - Manejo de los eventos enviados por los clientes, cada uno de ellos debe ser despachado al *Handler* correspondiente. Algunos de estos manejan un Thread propio según se consideró conveniente.
  - Observación de los cambios del juego (controlado por un Thread específico para cada mapa) y enviar las actualizaciones a los clientes correspondientes.
- Construcción de eventos entre los clientes y el servidor. Esto requirió coordinar con los otros integrantes la forma en la que los datos serían recibidos por los diferentes objetos.

1.3. Nicolas Aguerre

- Diseño y planteo de un conjunto de clases que sirvieran como primitivas para la construcción de todo el motor gráfico del juego (como una ventana, una textura, un texto, un sprite animado, un timer para eventos basados en tiempo, etc...). Estas clases en esencia serian *wrappers* de las estructuras de SDL.
- Implementación de una forma de almacenar toda la información relacionada a los gráficos del juego de forma organizada, y que permitiera agregar nuevos gráficos de forma sencilla, lo cual llevó al diseño de un conjunto de índices que contuvieran las rutas, y el correspondiente parseo de los mismos desde el programa.
- En relación al item anterior, la implementación de un gestor de *assets*, en el cual se mantuviera toda la información multimedia asociada al juego (texturas, sonidos y fuentes).
- Diseño y e implementación de la carga de mapas. Para esto, fue necesario el diseño de una clase que lograra la generación de un mapa a partir de el output del software Tiled.
- Construcción de *tilesets* apropiados a partir de los gráficos originales de Argentum Online, y otras fuentes de arte libre para juegos.
- Implementación de un mecanismo de abstracción de los tamaños y posiciones (cámara) del lado del cliente.
- Diseño e implementación de la interfaz gráfica de usuario, desde los dibujos para las mismas hasta la implementación dentro del juego.
- Implementación de la arquitectura del cliente necesaria para generar el flujo entre vistas, y para actualizar apropiadamente el estado del juego según las actualizaciones del servidor.
- Implementación de manejo de eventos de usuario del lado del cliente, e interacción con la interfaz gráfica.
- Implementación del sistema de sonidos.

2. Evolución del proyecto

TODO

3. Inconvenientes encontrados

TODO

4. Puntos pendientes

4.1. Features de juego pendientes

Algunas de las features pendientes son:

- Ataques a distancia y hechizos.
- Comportamiento de los NPCs amigables (Banquero, Sacerdote y Vendedor)
- Dinamizar la generación de monstruos a partir de un archivo de configuración en tiempo de ejecución (en este momento, la generación está hardcodeada en el codigo del servidor).
- Implementar mapa seguro (en este momento se admite el combate en cualquier mapa)
- Agregar clase manager para ciertas constantes que se usan a lo largo de las ecuaciones del juego, por ejemplo, en Player, ExperienceComponent, etc.
- Agregar mas mapas.

4.2. Features de UX pendientes

- Instalador mediante paquete de Debian.
- Generacion/restauracion de archivos de configuracion default desde el codigo.

4.3. Documentacion del trabajo practico pendiente

La documentación existente al momento de la primera entrega (07/07) es bastante escasa. Para la entrega final, se completaran:

- Manual de usuario, con screenshots de uso, mensajes de error y sus posibles soluciones.
- Documentación tecnica, con diagramas de clase de los modulos que componen el proyecto, especificación de como se crean mapas, como se indexan nuevos assets (texturas, sonidos y fuentes), como se modifica el spawneo de monstruos en el server a partir de los archivos de configuracion, etc.
- Manual de proyecto, agregando los bosquejos iniciales de la arquitectura del servidor, los diagramas finales de dicha arquitectura, y el *timeline* de progreso asentado por PivotalTracker.

5. Herramientas utilizadas

5.1. Sistema de control de versiones

Se utilizó el sistema de control de versiones **git**, y el gestor de repositorios online github. Este sistema de control de versiones ya venia siendo utilizado por los integrantes del grupo desde el comienzo de la materia, y resultó en verdad una pieza fundamental del desarrollo del presente proyecto.

5.2. Compilador

El compilador utilizado para el desarrollo fue GCC (y en concreto el compilador de C++, *G++*), bajo el estandar C++11.

5.3. Automatizacion del proceso de compilacion

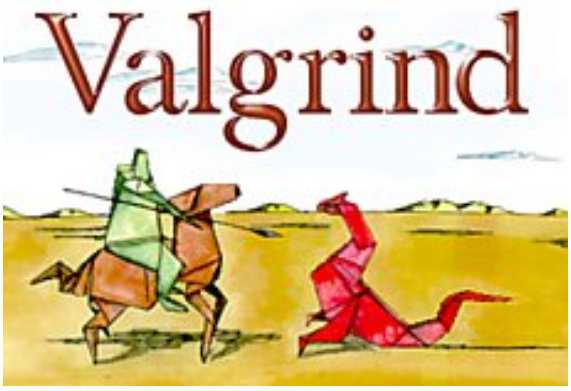
Para facilitar el proceso de compilación, se utilizaron dos herramientas diferentes: *CMake* y *Make*.
CMake resultó ser una herramienta maravillosa, siendo este el primer proyecto en el que cualquiera de los integrantes la usara. Fue utilizada tanto para automatizar la generación de un archivo MakeFile de compilación, como para automatizar la instalación/despliegue tanto del cliente como del servidor durante todo el desarrollo.

5.4. Debugging

El debugeo durante todo el proceso de desarrollo fue hecho utilizando las siguientes dos herramientas, cada una con su propósito:

- GDB, utilizado para debugeo general y seguimiento de la ejecución del código.

- Valgrind, utilizado para debugeo de memoria y verificación de pérdidas de memoria. Cabe recalcar que no fue posible utilizar demasiado Valgrind en el cliente, debido a que la biblioteca grafica utilizada (SDL) tiene leaks de memoria que escapan a nuestro alcance.



5.5. Diseño de mapas

Para el diseño de mapas, se utilizó el software *Tiled*, que es una herramienta que permite la construcción gráfica de mapas basados en tiles (baldozas).

5.6. Entorno de producción

Para simular el despliegue del servidor en un entorno de producción final, se utilizó una instancia de *Google Cloud* para desplegar el servidor.

Google Cloud

5.7. Generacion de documentacion

Para generar la referencia de clases, se formatearon los comentarios de los headers en formato Doxygen, herramienta que luego fue utilizada para generar la referencia del proyecto.

6. Conclusiones

TODO