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.
- Implementacion de los NPC's
- 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

El cliente y el servidor fueron desarrollados en paralelo durante toda la ejecucion del proyecto. A continuacion se ilustra una evolucion simplificada del mismo:

- **Semana 1:** Base grafica, Camara, Indexado. Base de arquitectura de threads para envio y recibo de mensajes. Base de comunicacion cliente-servidor.
- **Semana 2:** Sistema de entidades y componentes en el cliente. Empezada lógica del servidor. Definida y empezada la implementación de los eventos entre el cliente y el servidor. Completado movimiento del jugador en el servidor.
- **Semana 3:** Agregado envio de actualizaciones de informacion visual del servidor al cliente. Incorporado chat. Agregada persistencia. Comenzado parseo de comandos en el servidor. Conexion de multiples jugadores en simultaneo funcionando. Agregado manejo para quitar entidades tanto en el servidor como en el cliente. Agregados al cliente sprites de cascos, escudos, armaduras y armas, y coordinada dicha informacion con el servidor. Agregados susurros.
- **Semana 4:** Agregado nuevo tileset y mapa. Empezada la logica del orden de renderizado en el cliente. Refactors de logica en el servidor. Agregadas teletransportaciones entre mapas y en el mismo mapa, tanto en el cliente como en el servidor. Empezado el HUD en el cliente (barras de vida y boton de casteo). Comenzada logica de combate entre jugadores. Comenzado sistema de sonidos y agregada musica. Agregada vista de login.
- **Semana 5:** Agregado fullscreen y vista de creacion de personajes. Agregados iconos de items equipados en HUD. Implementado inventario. Implementados monstruos y NPCs agresivos en general. Implementado combate PvE. Agregado Loot y muerte en el cliente. Agregado revivir basico. Agregada docuemntacion con Doxygen. Agregados sonidos de combate. Comenzada documentacion.
- **Semana 6:** Agregados chequeos y handleo de errores. Agregados controles para hacer mas amigable la interfaz (desequipado con doble click). Implementadas diferencias entre razas y clases en el servidor para calcular stats y demas. Fixeados bugs al compilar con el compilador en modo pedantic, y realizados otros checks estaticos. Agregados nombres de entidades y texto de combate. Agregados NPCs amigables con sus comportamientos. Agregadas alertas de nombre inexistente, usuario ya online y nombre ya tomado. Implementado generador de paquete de debian para el cliente. Fixes de interacciones con NPCs amigables. Agregado manejo de hechizos tanto en el cliente como en el servidor.
- **Semana 7:** Creada github page para la documentacion. Fixes de performance, de memoria. Refactors. Agregado spawner de mobs agresivos mas prolijo en el server. Implementada teletransportacion del resucitar. Agregadas barras de vida. Agregadas limitaciones de FPS en el cliente, y posibilidad de activar/desactivar la vsync desde la configuracion sin recompilar. Implementada meditacion. Agregados detalles para hacer mas amigable la interfaz (nombre del item seleccionado, y resalte de item seleccionado en el inventario). Agregado panel de ayuda in-game. Terminada la documentacion. Hecho el trailer.

3. Inconvenientes encontrados

La parte mas difícil del desarrollo fueron las primeras semanas, al ser un proyecto mucho más extenso en comparación a cualquiera de los desarrollados hasta el momento por cualquiera de los integrantes, la perspectiva de su extensión fue difícil de visualizar. Sumado a esto, la definición de una arquitectura y diseño base que fueran suficientemente flexibles para lograr todas las funciones necesarias resultó particularmente desafiante, pero una vez estuvo pensado y armado, el agregado de funciones se volvió mucho más mecánico. Una vez definida la arquitectura base, la mayoría de los inconvenientes fueron mas bien implementativos:

- La implementacion de widgets de UI desde cero fue un poco molesta, pero no presento demasiados inconvenientes.
- El renderizado de TrueType fonts con SDL no es particularmente amigable, por lo que se termino optando por usar fuentes Bitmap para el texto in-game.
- Estabilidad del servidor al manejar diferentes tipos de *Race-Conditions*: nuevos clientes, desconexión de clientes, acciones y actualizaciones sobre un mismo mapa (y sus componentes).

4. Herramientas utilizadas

4.1. 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.

4.2. Compilador

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

4.3. Automatizacion del proceso de compilacion

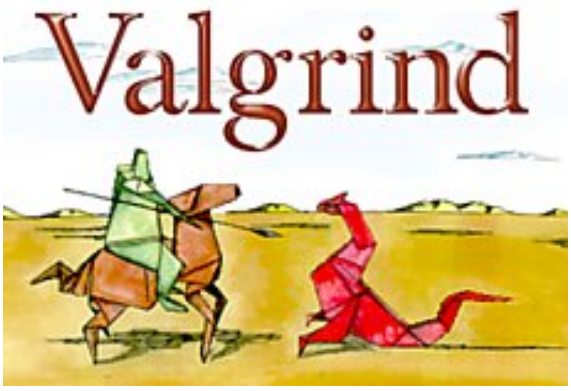
Para facilitar el proceso de compilación, se utilizaron dos herramientas diferentes: *CMake* y *Make*.
CMake resultó ser una herramienta maravillosa, siendo este el primer proyecto en el que cualquiera de los integrantes la usara. Fue utilizada tanto para automatizar la generación de un archivo MakeFile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

4.4. 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.



4.5. 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).

4.6. 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

4.7. 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.

5. Despliegue en Google Cloud

A modo de experimento, y para probar la robustez y portabilidad del servidor, se creó una instancia de Google Cloud en la cual el servidor fue desplegado (mediante clonado del repositorio y compilado remoto, a pesar de que nos hubiera gustado usar containers).

Nombre	Zona	Recomendación	Usada por	IP interna	IP externa	Conectar
aoserver-prod	southamerica-east1-c			10.158.0.3 (nic0)	35.198.39.89	SSH

```
nicolas@NicoDesktop ~ ssh nicomatex@35.198.39.89

Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-1030-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Jul 18 21:12:17 UTC 2020

  System load:  0.29              Processes:           135
  Usage of /:   45.2% of 9.52GB   Users logged in:     0
  Memory usage: 2%                IP address for ens4: 10.158.0.3
  Swap usage:   0%

 * "If you've been waiting for the perfect Kubernetes dev solution for
   macOS, the wait is over. Learn how to install Microk8s on macOS."

   https://www.techrepublic.com/article/how-to-install-microk8s-on-macos/

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

5 packages can be updated.
1 update is a security update.


Last login: Tue Jul 14 21:05:08 2020 from 186.141.135.119
```

```
nicomatex@aoserver-prod:~/repo/server/build$ ./aoserver
Dispatcher: starting..
ClientDropHandler: starting..
ChangeMapHandler: starting..
ClientInitializeHandler: starting..
CommandHandler: starting..
CreationHandler: starting..
MapManager: creado mapa: "Paseo Colon"
MapManager: creado mapa: "Reserva Puerto Madero"
MapManager: creado mapa: "Facultad de Ingenieria"
ServerManager: creating session for map id: 2
ServerManager: creating session for map id: 0
ServerManager: creating session for map id: 1
```

6. Conclusiones

Este ha sido sin duda el proyecto de mayor complejidad y envergadura que cualquiera de los integrantes del grupo haya realizado. Fue una experiencia de desarrollo mucho mas completa que cualquier trabajo practico anterior, y aportó mucho a nuestras habilidades como programadores, tanto a nivel proyecto, como en experiencia con el uso de C++. Es gratificante poder ver el juego ya terminado y funcionando, y mas aun poder compartirlo y jugarlo con personas ajenas al desarrollo.