Argentum - Manual de Proyecto

[7542/9508] Taller de Programación
Primer cuatrimestre de 2020

FRITZ, Lautaro Gastón	102320
FABBIANO, Fernando	102464
NOCETTI, Tomás	100853

Link al repositorio de GitHub:
https://github.com/tomasnocetti/taller-trabajo-final

Índice

1. División de tareas y evolución

Desde un primer instante se planteo una arquitectura y orden de proyecto para lograr ser lo mas productivos y eficientes posibles considerando que el tiempo dado es limitado. En la primera semana luego de un análisis integro del TP se resolvieron los siguientes Roles:

- Lautaro (Cliente)
  - Interfaz Gráfica: Animaciones, texturas, textos y sonidos.
  - Armado de mapas.
  - Búsqueda de recursos para el armado del juego.
- Fernando (Servidor)
  - Lógica dura del juego (features) con todo lo que eso involucra.
  - Arquitectura servidor.
- Tomás (Cliente/Servidor)
  - Arquitectura cliente/servidor.
  - Arquitectura multi-jugador.
  - Persistencia y manejo de archivos.

Basado en este esquema en general las responsabilidades se mantenían constantes para los primeros dos integrantes, quienes se dedicaban exclusivamente a esos sistemas, mientras el ultimo iba moviéndose para compensar. De esta manera se logró avanzar a un ritmo parejo, sabiendo que las demandas gráficas iban a ser mas fuertes al principio y las demandas lógicas mas fuertes al final. En este sentido se sabia entonces que el cronograma propuesto iba a ser difícil de cumplir debido a que las responsabilidades diferían a las de la propuesta original.

En base a ese esquema se muestran el cronograma original y el final.

	Alumno 1 **Servidor - Modelo**	**Alumno 2** **Modelo - Cliente**	**Alumno 3** **Modelo - Cliente**
Semana 1 (09/06/2020)	Carga de mapas. Lógica de movimiento de los personajes y NPC.	Mostrar una imagen. Mostrar una animación. Mostrar ambas en un lugar fijo o desplazándose por la pantalla (movimiento).	Reproducir sonidos, musica. Mostrar texto en pantalla.
Semana 2 (16/06/2020)	Lógica de ataque, defensa, drops tanto de jugadores como de NPCs) Razas y clases.	Mostrar todo el mapa, incluyendo varios tipos de terrenos distintos, edificios y unidades (jugadores y NPC).	Mostrar la interfaz gráfica (barra de experiencia, items, vida)
Semana 3 (23/06/2020)	Lógica de las ciudades, “fair play”. Compra, venta y otros comandos. Experiencia y niveles.	Mostrar los objetos (drop). Interacción por parte del usuario.	Mini-chat tanto para leer los mensajes como para escribirlos. Mensajes dirigidos a un NPC y a un jugador.
Semana 4 (30/06/2020)	Sistema de comunicación (cliente - servidor) Servidor completo.	Cliente completo.	Pantalla de login. Persistencia y configuración.
Semana 5 (07/07/2020)	- Pruebas y corrección sobre estabilidad del servidor.	- Pruebas y corrección sobre estabilidad del cliente.	- Pruebas y corrección sobre estabilidad del cliente.

	- Detalles finales y documentación preliminar	- Detalles finales y documentación preliminar	- Detalles finales y documentación preliminar
Primera Entrega el dia 07/07/2020			
Semana 6 (14/07/2020)	- Correcciones sobre Primera entrega - Testing y corrección de bugs - Documentación	- Correcciones sobre Primera entrega - Testing y corrección de bugs - Documentación	- Correcciones sobre Primera entrega - Testing y corrección de bugs - Documentación
Semana 7 (21/07/2020)	- Testing - Documentación - Armado del entregable	- Testing - Documentación - Armado del entregable	- Testing - Documentación - Armado del entregable
Entrega Final el dia 21/07/2020			

Figura 1: Cronograma Original

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7
Persistencia, Carga de Mapas y Configuración	- Pruebas persistencia con Msg Pack. - Pruebas con Tiled para creación de Mapas	- Lógica de parseo completo del mapa del lado del cliente con JsonCPP.	- Mapa provisorio con spawn points de NPC's y limites	- Armado de mapa definitivo con capas de colisiones, ciudades y NPC's spawn points	- Persistencia de tamaño fijo. - Se envia mapa desde el servidor. - Archivo de conf. JSON		
Arquitectura de comunicacion cliente-servidor		-Diseño de protocolo, implementacion cliente y servidor. Arquitectura para el movimiento	- Se termina la arquitectura con distintos tipos de Instruciones y Respuestas		- Autenticación y creacion de personaje		
Logica de Juego	- Manejo basico y provisorio de movimiento desde el cliente	- Lógica de movimiento, colisiones y ataque contra NPC's. Sin comunicación cli-serv.	- Movimiento npcs, movimiento jugadores, colisiones y logicas varias integradas en Cron	- Logica de resucitar, recuperar vida, meditar. Distintas armas y armaduras. Respawn NPC y ciudades, niveles.	- Logica comerciante, banquero, sacerdote, drops con sus respectivos comandos. Fairplay y Chat	- Logica de sonidos. - Desequipar item.	
Arquitectura Servidor	- Modelo provisiorio de arquitectura multijuador	- Client aceptor y Game server. - Client Proxy multihilo para lectura y escritura. - Propagación de modelos	- Diseño de arquitectura de Cron de Juego con comunicacion Modelo	- Pruebas de estabilidad, memoria y cierre del juego	- Arquitectura para constantes, soft stop y logger.		- Estabilidad y mejoras para juego multijuador. - Constant fame rate GameCron
Arquitectura cliente	- Modelo MVC para el cliente. - Investigacion SDL2. - Armado de entorno de trabajo	- Server Proxy multihilo para lectura y escritura. - Sistema de respuestas	- Sistema ECS cliente - Analisis de memoria y estabilidad	- Manejador de texturas y fuentes			- Mejoras en el manejo de CPU y memoria
Interfaz Grafica Mapa y Jugador	- Renderizado de personaje.y animacion	- Sistema de camara sobre mapa dinamico, jugador con movimiento y elementos como arboles. - Sin colisiones, movimiento cliente.	- Adaptacion de movimiento, visual multijugador y multi NPC animados, Ataque de jugador	- Armas y equipo con animaciones, pocimas y elementos varios visuales.	- Elementos restantes de armas y equipo. - Musica de ambiente. - Drops	- Logica de sonidos.	- Mejoras en animaciones, mapas multilayer y mejoras en general de la interfaz
Inventario, Barras y Chat, Creacion y Login		- Barras de vida, mana y experiencia funcionales. - Distintos tipos de jugadores	- Inventario con correspondiente manejo de eventos	- Interfaz del chat con logica de scroll y ingreso de comandos.		- Interfaz final de creacion y login.	
Manuales, Documentacion y Entorno					- Manual del usuario - Instalador	-Manual del Proyecto	- Manual Tecnico - Manual Tecnico

Tomás Fernando Lautaro

Figura 2: Cronograma Final

2. Inconvenientes encontrados

Como todo proyecto en su transcurso fueron apareciendo bugs e inconvenientes que nos forzaron a repensar el como se estaba atacando determinada problemática. En esta sección solo se mencionaran los que hayan resultado mas interesantes sin entrar en tecnicismos.

2.1. Persistencia de tamaño fijo. ¿Qué tan fijo?

Una de las condiciones del trabajo determinaba que el estado del jugador debía ser preservado en bloques de tamaño fijo en disco. De esta manera se lograba acceder a índices específicos para buscar la información de dicho jugador sin tener que cargar toda esa información en memoria. El inconveniente apreció con el uso de la librería de serialización MsgPack cuyo uso fue sugerido por la cátedra. El problema de esta magnifica librería recaía en que la serializacion de dos estructuras de datos que suponían ser iguales, no lo eran. Esto sucedía debido a que el contenido de las estructuras no era igual. Ejemplificando:

```
int a = 1000; // PACK(a) == cd 03 e8
int b = 10;   // PACK(b) == 0a
```

Nada garantiza que los tamaños de esas estructuras en binario sean iguales entonces no era una opción valida. Cuales eran las opciones ?

- Armar la serialización nosotros, lo cual resultaba muy poco práctico e iba a llevar mucho tiempo.
- Calcular un tamaño mayor lo que garantizaba tamaños fijos pero generaba un desperdicio de espacio.
- Buscar otra librería.

2.1.1. ¿Cómo se resolvió?

Al final luego de una investigación sobre la documentación de MsgPack se vió que existían dos maneras de serializar, una invasiva (Macros), que era la que nosotros conocíamos, y una no invasiva (Adaptors). Esta ultima nos daba mas control sobre las estructuras y nos permitía utilizar funciones propias de la librería que sí nos garantizaban tamaños fijos. Si bien en comparación su uso era mas complejo, resultaba mucho mas sencillo que las otras opciones.

2.2. Lógica de movimiento. ¿Cliente ó Servidor?

Una de las primeras problemáticas que surgieron, en parte porque era uno de los primeros aspectos lógicos que se iban a tener que resolver, fué la lógica del movimiento. Entendiendo que ninguno de los integrantes tenía conocimiento de como armar una arquitectura multijugador estábamos desorientados de en donde debía residir dicha lógica. Si esta estaba del lado del cliente era posible que existiese un problema de seguridad, un jugador podría enviarle sus coordenadas al servidor y "falsificar su posición", al mismo tiempo surgía el inconveniente de que podría haber jugadores que se movieran mas rápido que otros. Si bien era lo mas sencillo en un principio, iba a traer problemas a futuro. La otra opción era que el servidor controlase el movimiento de todos los jugadores, pero cómo era esto posible?

2.2.1. Un Cron para dominarlos a todos

Si bien el jugador desde el lado del cliente tiene que poder ser autónomo en su juego, lo que se le permitía era que envié al servidor una "dirección de movimiento". De esta manera el servidor recibía dicha instrucción y en un hilo aparte, autónomo del que controlaba la lógica en si, se procesaba, cada determinado tiempo, todas las direcciones de movimiento de todos los jugadores y se enviaban coordenadas al hilo principal para que mueva al jugador. Esto garantizaba consistencia

en el movimiento de todos los jugadores y seguridad. Finalmente este Cron no solo se usó para la lógica de movimiento, sino también para los NPC's y distintas acciones del juego que necesitaban periodos de tiempo para ejecutarse. La regla era, si implicaba tiempo –>CRON.

2.3. Un pequeño cambio de paradigma, un gran cambio para el código

Ya en las ultimas semanas de desarrollo cuando la demanda lógica empezó a aumentar y los distintos features se hacían mas sencillos de realizar, llegó el momento de migrar las constantes que requería el servidor a un archivo de configuración. De aquí surgió la necesidad de revisar la forma en la que estábamos manejando los items del juego. Llamamos items a los objetos/elementos a los que el juego le permite al jugador interactuar. Ej: armas, pociones, escudos, etc. Estos items son comunes a muchos aspectos lógicos, los comerciantes y los sacerdotes venden items, los banqueros guardan items, los jugadores equipan items, etc. Bajo este paradigma vimos que la manera en la que estábamos haciendo las cosas no era la correcta, el código se repetía en bloques por varias partes del proyecto y se nos hacia complejo trabajar con Macros para los distintos aspectos lógicos. Pero ahí llegó el dilema, estando en un punto avanzado del desarrollo, conviene realizar una refactorización de esta magnitud ?

2.3.1. ¿Conviene, o no conviene?

Luego de un análisis y de estimaciones, vimos que la refactorización valía la pena. Si bien iba a consumir tiempo que no estaba dentro de los planes dedicarle, asumimos que una vez terminado, el desarrollo de varios de los features que nos faltaban se iba a acelerar notablemente y el código iba a quedar mejor armado. Es en este punto que confirmamos que la decisión fué acertada.

2.4. Siempre se puede optimizar un poco más

Ya cerca de la instancia de entrega final, con correcciones de la pre-entrega, los items mas preocupantes venían de la mano con usos intensivos de CPU por parte del cliente y LAG con múltiples jugadores por parte del servidor. Pero cómo ? Si a lo largo del trabajo se presto particular atención a las optimizaciones posibles, si se optimizo la transferencia de información para no saturar la red y se buscó en todo momento lograr tener un servidor que orquestre y sincronice los jugadores. Sabíamos que algunas de las arquitecturas que estábamos usando no eran óptimas, aunque eran las recomendadas por una cuestión de complejidad y tiempos, pero carecía de sentido que estas problemáticas apareciesen bajo poca demanda. Entonces la pregunta era dónde estaba el problema.

2.4.1. La pregunta es dónde

La realidad es que la arquitectura no estaba del todo cerrada y aquí el análisis se parte en 2.

LAG - El LAG era producto de que el hilo del cliente no estaba dando abasto a procesar toda la información que le llegaba del servidor. Nuestro modelo planteaba en todo momento que el cliente tenga un snapshot acotado del modelo entero del juego, el problema venía en que cada instrucción que el servidor procesaba, propagaba esos snapshots saturando al cliente. La solución vino en realizar un "throttling"por parte del servidor que permita la actualización de los modelos y cada X tiempo propague dicha información al cliente.

CPU - Aquí se complicó un poco mas, el uso intensivo de CPU se originaba por iteraciones muy extensas que demandaban un uso intensivo (EJ iterar 32000 elementos del mapa para renderizarlo cada 25ms). El problema era que el uso de CPU variaba entre los integrantes, haciéndose difícil ver mejoras de manera eficaz. Se tomo como patrón a uno de los integrantes y en base a eso se decidía si la mejora servía o no.

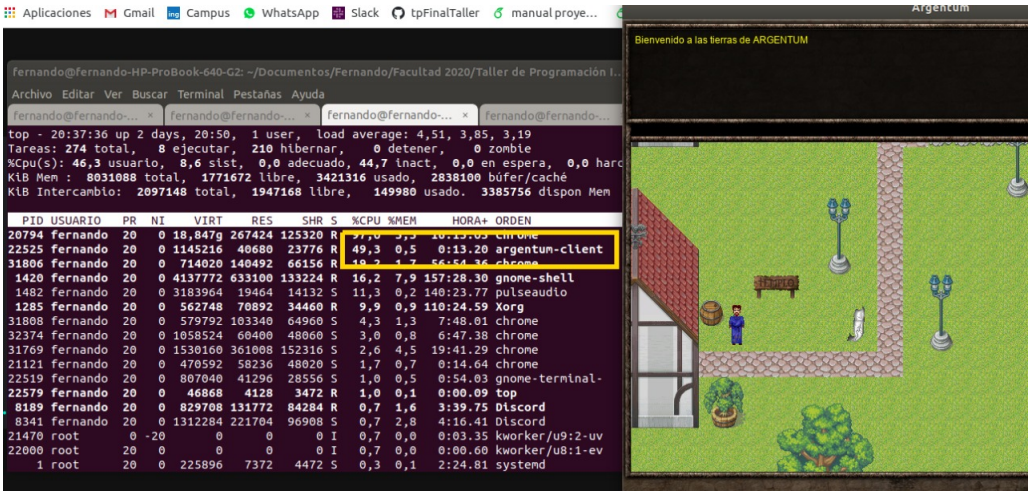

Figura 3: Detalle de uso del cliente, CPU al 50 por ciento

Fueron varios los intentos y las mecánicas que se usaron para reducir al mínimo dichas iteraciones, desde accesos a porciones especificas del arreglo, hasta cambios en estructuras auxiliares como mapas.

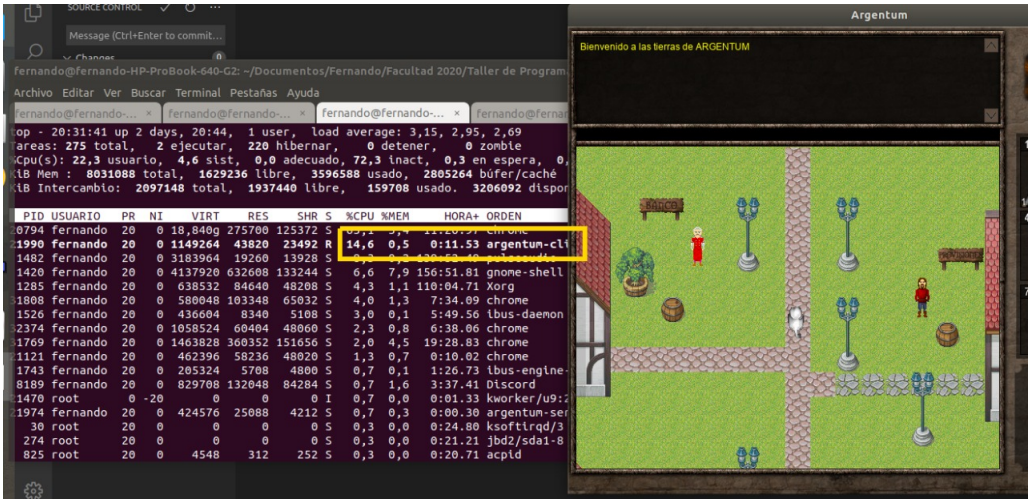

Figura 4: Detalle de uso del cliente, CPU al 25 por ciento

La realidad es que según algunas investigaciones realizadas, SDL2, la librería que se encarga del manejo de la interfaz gráfica, es bastante demandante en su uso de recursos por lo que nuestras mejoras tuvieron un techo.

3. Herramientas

A continuación se listan las herramientas que hicieron el desarrollo mas sencillo.

3.1. IDE

Para el desarrollo total del proyecto se decidió utilizar el editor VS Code debido a que disponía de una gran facilidad a la hora de Debuggear código C++. VS Code permite agregarle una serie

de complementos que agregan funcionalidades como Linters y Formatters que hacen el desarrollo mas ameno. Algunas de estas extensiones:

- https://github.com/microsoft/vscode-cpptools
- https://github.com/twxs/vs.language.cmake

3.2. Makefile

Se procedió a realizar un Makefile para el trabajo, basado en uno provisto por la cátedra, que facilito el separar el desarrollo en dos partes, cliente y servidor, logrando de esta manera desacoplar problemas y disminuir los tiempos de compilación para las personas que trabajaban específicamente en una de las partes. Dicho Makefile se encuentra disponible en el repositorio del proyecto.

3.3. GitHub

Nuestro gran aliado, Github (o mejor dicho git) nos permitió mantener una linea de código limpia sobre el branch Master. Luego cada integrante trabajó sobre 1 ó mas branches procurando de esta manera avanzar con sus tareas sin interferir en el desarrollo de los demás. En casos como los mencionados en secciones anteriores, donde los cambios afectaban varias partes del proyecto, se trabajaba sobre un nuevo branch Dev que permitía no alterar Master hasta que el trabajo este completo.

3.4. Tiled

Un editor de mapas open-source de fácil uso y bien documentado. Esta fue la herramienta que se uso para generar los primeros mapas de prueba y los mapas definitivos. El parsing del mapa fue desarrollado por el equipo viendo que no había ninguna librería en la industria que cumpliese con nuestros requerimientos.

3.5. Photoshop

Editor de fotografía que se utilizó para crear/modificar las distintas interfaces visuales que presenta el proyecto. Los PSD fueron obtenidos de manera gratuita del siguiente sitio https://graficosargentum.jimdofree.com/tutoriales/.

3.6. Overleaf

Toda la documentación del presente proyecto se realizo en el formato LaTEX a traves de la herramienta Overleaf.

4. Conclusiones

El trabajo fue extenso y fueron muchas las situaciones en las que el equipo se tuvo que sentar a pensar y debatir cual era el mejor camino. Se trató de manetener una comunicación constante, mas considerando que el 1er cuatrimestre 2020 fue uno especial. Otro aspecto en lo que se puso foco fue el de hacer las cosas una vez pero bien, aunque esta lleve un poco más de tiempo. El equipo se encuentra muy conforme con el desempeño obtenido y con la integración de conocimientos realizada.