Manual de Proyecto

February 23, 2021

Germán Rotili	german.rotili@gmail.com
Joaquin Lopez Saubidet	jlopezs@fi.uba.ar
Luciano Ortiz	lnortiz@fi.uba.ar
Andres Tomas Visciglio	avisciglio@fi.uba.ar

1 Enunciado

Wolfenstein 3D

Ejercicio Final

Objetivos	● Afianzar los conocimientos adquiridos durante la cursada. ● Poner en práctica la coordinación de tareas dentro de un grupo de trabajo. ● Realizar un aplicativo de complejidad media con niveles aceptables de calidad y usabilidad.
Instancias de Entrega	**Primera Entrega:** clase 13 (09/02/2021). **Segunda Entrega:** clase 15 (23/02/2021).
Temas de Repaso	● Aplicaciones Cliente-Servidor multi-threading. ● Interfaces gráficas ● Manejo de errores en C++
Criterios de Evaluación	● Criterios de ejercicios anteriores. ● Construcción de un sistema Cliente-Servidor de complejidad media. ● Empleo de buenas prácticas de programación en C++. ● Coordinación de trabajo grupal. ● Planificación y distribución de tareas para cumplir con los plazos de entrega pautados. ● Cumplimiento de todos los requerimientos técnicos y funcionales. ● Facilidad de instalación y ejecución del sistema final. ● Calidad de la documentación técnica y manuales entregados. ● Buena presentación del trabajo práctico y cumplimiento de las normas de entrega establecidas por la cátedra (revisar criterios en sitio de la materia).

Índice

Introducción

El presente trabajo consistirá en hacer un remake del mítico Wolfenstein 3D, creado por ID Software en 1992, considerado uno de los juegos fundadores del género First Person Shooter.[1]

El jugador hará el papel de William "B.J." Blazkowicz, un espía de los Aliados infiltrado en el castillo Wolfenstein lleno de Nazis.

En esta remake el juego será multijugador competitivo donde cada jugador será B.J. y verá al resto de los jugadores como oficiales Nazis.

Descripción

Armas

Hay 5 tipos de armas en el juego:

 **Cuchillo**: es el arma por default cuando el jugador se queda sin balas. El ataque se realiza cada vez que el jugador presiona la tecla de disparo. A diferencia del resto de las armas está solo produce daño si el jugador está al lado del enemigo.

 **Pistola**: solo puede disparar de a una bala a la vez cada vez que el jugador presione la tecla de disparo.

 **Ametralladora**: dispara 5 balas por ráfaga. El jugador puede continuar disparando ráfagas mientras mantenga presionada la tecla de disparo. En este caso las ráfagas tendrán una frecuencia de 0.3/sec (no hay un disparo perfectamente contiguo)

 **Cañón de cadena**: dispara 1 bala cada 0.1 secs en una ráfaga continua mientras el jugador presione la tecla de disparo.

 **Lanzacohetes**: a diferencia de otras armas, lanza un proyectil que es visible y que se mueve por el escenario hasta impactar en un objeto o enemigo, causando daño a todos los enemigos que este en un rango incluyendo al mismo jugador. Cada disparo gasta 5 balas. Deberá haber una animación para la explosion del proyectil.

Cada arma (excepto el cuchillo y el lanzacohetes) tiene una precisión, una probabilidad de acertar en el blanco y producir un daño en el enemigo.

La precisión también es modificada según la distancia y el ángulo entre la posición del enemigo y la dirección de disparo.

Un enemigo que está a mayor distancia recibirá un daño menor que un enemigo que esté más cerca.

Un enemigo que esté exactamente enfrente del jugador recibirá mayor daño que uno que se encuentre un poco más a un costado.

El daño producido será un número aleatorio entre 1 y 10 por cada impacto de bala.

En el caso del cuchillo este causara daño siempre que se esté en el rango de alcance y su daño también será aleatorio entre 1 y 10.

El proyectil del lanzacohetes se mueve en línea recta y al impactar genera también un daño aleatorio pero inversamente proporcional al punto de impacto: enemigos cercanos reciben mayor daño.

Cuando el jugador se queda sin balas, su arma cambia al cuchillo. Si adquiere balas su arma vuelve a ser el arma previa.

El jugador puede tener más de una arma pero solo podrá usar 1 de ellas a la vez y podrá elegir cual con algún atajo del teclado.

Enemigos

Hay 5 enemigos en el juego. Cada jugador se verá a sí mismo como B.J. y verá a los otros jugadores como soldados Nazis.

Dependiendo del arma que tenga cada jugador estos se verán de una u otra manera.

 **Perro**: cuando el jugador tenga equipado un cuchillo.

 **Guardia**: cuando el jugador tenga equipada una pistola.

 **SS**: cuando el jugador tenga una ametralladora.

 **Oficial**: cuando el jugador tenga un cañón de cadena.

 **Mutante**: cuando el jugador tenga equipado un lanzacohetes.

Items

En el escenario habrá una serie de ítems en el piso que el jugador podrá recolectar.

 Comida: recuperan parte de la vida del jugador unos 10 puntos.
 Kits médicos: recuperan unos 20 puntos de la vida del jugador.
 Sangre: recupera 1 punto de vida solo si el jugador tiene menos de 11.
 Balas: suman 5 balas
 Ametralladora: arma que un jugador puede tomar.
 Cañón de cadena: arma que un jugador puede tomar.
 Lanzacohetes: arma que un jugador puede tomar
 Tesoros: cruces, copas, cofres y coronas, suman 10, 50, 100 y 200 puntos cada uno.
 Llaves: algunas puertas solo pueden abrirse si se tiene una llave.

Los ítems son recolectados automáticamente al pasar por encima de ellos.

Hay unas excepciones:

Ni la comida ni el kit médico son recolectados si el jugador ya tiene todos los puntos de vida. La sangre no es recolectada si el jugador tiene 11 puntos o más.

Las balas no son recolectadas si el jugador ya tiene el máximo de balas permitido.

Ninguna arma es recolectada si el jugador ya tiene esa arma.

Motor 3D: ray casting

Así como el juego original tenía un motor 3D propio, en el presente trabajo se deberá implementar el motor 3D usando la técnica de ray casting.

La pantalla que el jugador ve se descompone en tiras verticales de 1 pixel de ancho. Dada una pantalla de 320 (ancho) por 200 (alto) la pantalla tendrá 320 tiras verticales de 1 pixel de ancho y 200 de largo.

Por cada tira vertical se proyecta un rayo saliente del jugador hacia la dirección a la que mira, de ahí el nombre ray casting.

Brevemente, el juego modela los escenarios como un mapa 2D dividido en una cuadrícula.

La pared en donde cada rayo impacta determina que imagen y que parte de esa imagen se debe renderizar para dicha tira vertical.

La distancia recorrida por el rayo determina luego el escalado para que las paredes más alejadas se vean más chicas dando así un efecto 3D.

Como el proceso de ray casting hay que repetirlo constantemente (30 veces por segundo), el motor del Wolfenstein tiene una serie de simplificaciones y optimizaciones.

El escenario está dividido en celdas, como si fuera una matriz.

Una celda puede estar ocupada por una pared. En la vida real las paredes son "planos" con muy poco espesor. En el juego una pared es un cuadrado (o cubo) que ocupa la totalidad de la celda.

Para detectar si un rayo colisiona o no con una pared deberíamos chequear todos los puntos que componen el rayo desde que sale del jugador hasta que encontremos la primera pared.

Algo super costoso.

Con la simplificación de las celdas ahora solo es necesario chequear las intersecciones con las celdas y no los puntos entre medio. Por ser cuadrados, hay como mucho 2 puntos por celda y sus posiciones pueden ser determinadas con simple trigonometría.

Suena confuso?

Hay muy buenos recursos online que describen esta técnica.

Es importante ver el siguiente material: la técnica de ray casting es explicada por 3 personas distintas y les permitirá tener una idea más clara.

Lode Vandevenne describe muy bien el proceso aunque en lo personal no me quedó clara la parte de cómo determina los puntos de cada rayo.[2]

F. Permadi por el otro lado describe el mismo proceso usando más trigonometría. En combinación con el trabajo de Lode se puede entender bastante bien el cómo funciona.[3]
También describe features más
avanzados como saltos y transparencias que no son necesarios en el presente trabajo.

Daniel Shiffman tiene un video explicando toda la técnica. No ahonda en detalles sin embargo y no hace uso de las simplificaciones originales del motor del Wolfenstein.[4]

Consejo: entender los artículos pero luego de resolver los problemas con papel y lápiz, paso a paso. En serio. El plagio no es una opción.

Escenario

Los escenarios estarán compuestos por habitaciones y pasillos. Además de las obvias paredes, en las habitaciones podrá haber objetos que obstruyan el paso como las mesas y los barriles y objetos que no, como manchas de agua o sangre en el piso.


Habrá también puertas. Estas se abren cuando un jugador se acerque y presione la tecla para abrirla. Luego de un tiempo, las puertas se cierran solas.


Ciertas puertas requerirán una llave para poder ser abiertas. Cuando el jugador abra la puerta usando una llave, la llave desaparece y esta puerta podrá ser abierta y cerrada por cualquier otro jugador sin ninguna llave.

Las paredes pueden ser de diferentes tipos siendo todas igualmente funcionales.

Ciertas paredes, sin embargo son “falsas”. El jugador podrá “mover” la pared haciéndola desaparecer y habilitando el paso presionado la misma tecla para abrir una puerta.

Estos son los famosos pasadizos secretos.

Vida y muerte

Cada jugador arranca el juego en un lugar específico determinado por el escenario o mapa, con una pistola y 8 balas.

Cada vez que un jugador muera dejará en su lugar el cadáver y su arma (salvo que tenga una pistola). Además dejará 10 balas en el piso y si tuviese tambien una llave.

El jugador muerto revivirá y podrá seguir jugando. Reaparecerá en el mismo lugar en el que arrancó el juego y tendrá como arma una pistola con 8 balas.

Un jugador podrá revivir hasta 2 veces. Luego no podrá continuar jugando.

Partida

Las partidas son multijugador. Cada nivel o escenario tendrá un número de jugadores posibles aunque se podrá comenzar la partida con menos de ellos.

Cada jugador se conectará al servidor y deberá poder elegir entre crear una partida nueva (donde elegirá el nivel) o unirse a una ya existente.

Una vez que los jugadores se hayan unido, se podrá dar por comienzo la partida. Una vez que la partida comenzó no se puede sumar nadie más.

La partida termina cuando todos salvo 1 jugador mueren (y no reviven) o por falta de tiempo.

Al finalizar la partida se deberá mostrar un top 5 de los jugadores con más enemigos matadas, más puntos por tesoros y más balas disparadas.

Integración con Lua

El juego podrá poner enemigos adicionales controlados por la computadora (AI). En vez de programarlos en C++, estos se deberán programar en Lua.

El servidor deberá poder cargar módulos escritos en Lua y exponer a ellos una representación del mapa y enemigos con el fin de que el módulo pueda tomar una decisión de que hacer (moverse, disparar). Dichas acciones son luego tomadas por el servidor y ejecutadas como si se tratase de un jugador más.

En resumen, todo lo que un jugador pueda hacer, un módulo de Lua debería poder hacerlo.

Se pide una integración y 1 solo modulo de Lua que sirva de ejemplo. No es necesario programar un módulo de AI completo.

Interfaz de usuario

El juego muestra el escenario, objetos, paredes y a otros jugadores desde una perspectiva de primera persona.

El jugador puede moverse, abrir puertas y disparar usando el teclado.

La interfaz debe mostrar también el arma equipada, los puntos de vida actuales, los de tesoro y las balas.

Según el nivel de vida se deberá mostrar una u otra cara de B.J.

El cliente debe poder mostrar el juego en fullscreen y en modo ventana siendo el modo definido en el archivo de configuración. En ambos casos la resolución de la pantalla también será determinada por archivo de configuración y serán valores típicos como 320x200 o 640x400

La imagen esta a efectos ilustrativos. No se requiere que la UI sea exactamente igual a esta.

Configuración

Todos los valores numéricos (constantes numéricas) deben venir de un archivo de configuración de texto YAML[5]: cantidad de balas máxima, vida, frecuencia de disparo, etc.

Tener los parámetros en un solo lugar les permitirán hacer modificaciones al juego si este está desbalanceado y hace que el juego no sea divertido.

Un archivo de configuración le permitirá a ustedes hacer "*ajustes*" más finos sin recompilar y al docente le permitirá cambiar ciertos valores para facilitar la corrección (por ejemplo se puede poner vida infinita)

Sonidos

Como todo juego se debe reproducir sonidos para darle realismo a los eventos y acciones que suceden[6]:

- Cuando hay un disparo se debe emitir el sonido característico.
- Cuando hay una muerte se debe emitir un sonido acorde.
- Cuando una puerta se abre o cierra.

Si la cantidad de eventos que suceden es muy grande, algunos sonidos deben ser evitados para no saturar al jugador con tanta información.

El volumen de los ruidos deberá estar modulado por la distancia: eventos cercanos producirán sonidos o ruidos más fuertes.

Musica

Se debe reproducir una música de fondo.

Animaciones

Las puertas, los jugadores, los disparos, todo que sea dinámico tendrá que ser animado [7][8][9].

Aplicaciones Requeridas

Servidor

El juego a implementar es multijugador con una arquitectura cliente-servidor. El servidor deberá recibir por parámetro la ruta a un archivo de configuración que contendrá:

- Puerto donde escuchar
- Parámetros del juego: todos los valores numéricos descritos en el presente enunciado deben ser configurables desde un archivo de texto. Esto es ***esencial*** para optimizar la jugabilidad y balancear las fortalezas y debilidades de las unidades sin tener que recompilar el código.

La definición del protocolo de comunicación entre el cliente y el servidor queda a libre elección. Se requiere sin embargo que sea binaria (no puede ser texto) y no pueden usarse librerías de terceros.

Se debe soportar múltiples partidas a la vez.

Cliente

Es el elemento central que interactúa con el jugador. Debe poder mostrarle una pantalla de login para definir a qué servidor quiere conectarse, si quiere crear una partida o unirse a una.
Ya iniciado el juego, debe ofrecer una interfaz rica de acuerdo a los requerimientos pedidos.

Editor

Junto con la aplicación cliente-servidor se deberá entregar una tercer aplicación: un editor de niveles o escenarios.

Se deberá poder diseñar los mapas colocando las paredes, puertas, tesoros y otros objetos. Además se deberá poder indicar en donde arrancará cada jugador.

El editor deberá poder abrir y guardar los mapas. Estos deberán estar en formato YAML.

Los mapas pueden ser más grandes de los que el editor puede mostrar en pantalla por lo que se tendrá que soportar scrolling.

Además deberá implementarse point and click y drag and drop como formas para editar el mapa (donde mejor convenga para el usuario).

Distribución de Tareas Propuesta

Con el objetivo de organizar el desarrollo de las tareas y distribuir la carga de trabajo, es necesario planificar las actividades y sus responsables durante la ejecución del proyecto. La siguiente tabla plantea una posible división de tareas de alto nivel que puede ser tomada como punto de partida para la planificación final del trabajo (*las fechas marcan el día en que la semana se completa y se deberían tener los temas terminados*):

	Alumno 1 - 4 Servidor - Modelo	Alumno 2 Modelo - Cliente	Alumno 3 Editor - Cliente
Semana 1 (24/11/2020)	Carga de mapas. Lógica de movimiento de los personajes (colisión con las paredes y otros objetos).	Mostrar una imagen. Mostrar una animación. Ecuación de los rayos (donde intersecciona con cada celda)	Reproducir sonidos, música. Mostrar texto en pantalla. Draft del editor.
Semana 2 (01/12/2020)	Lógica de las partidas. Lógica del ataque. Modulo IA basico.	Vista 3D las paredes usando ray casting. Sin texturas ni objetos ni otros jugadores	Editor basico
Semana 3 (08/12/2020)	Sistema de comunicación (cliente - servidor) Modulo IA completo.	Vista 3D incluyendo las texturas de las paredes, objetos y jugadores.	Editor completo.
Semana 4 (15/12/2020)	Servidor completo. Configuración.	Cliente completo.	Pantalla de login y de partidas.
Semana 5 (09/02/2021)	- Pruebas y corrección sobre estabilidad del servidor. - Detalles finales y documentación preliminar	- Pruebas y corrección sobre estabilidad del cliente. - Detalles finales y documentación preliminar	- Pruebas y corrección sobre estabilidad del cliente. - Detalles finales y documentación preliminar
Primera Entrega el día 09/02/2021			
Semana 6 (16/02/2021)	- 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 (23/02/2021)	- Testing - Documentación - Armado del entregable	- Testing - Documentación - Armado del entregable	- Testing - Documentación - Armado del entregable
Entrega Final el día 23/02/2021			

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema se debe realizar en ISO C++11 utilizando librerías *SDL y/o Qt*.
2. Con el objetivo de facilitar el desarrollo de las interfaces de usuario, se permite el uso de *QtDesigner* u otro editor de interfaces gráficas.
3. Se debe usar una librería de parsing YAML. No se puede implementar un parser propio.
4. Es condición necesaria para la aprobación del trabajo práctico la entrega de la documentación

mínima exigida (consultar sitio de la cátedra). Es importante recordar que cualquier elemento faltante o de dudosa calidad pone en riesgo la aprobación del ejercicio.

5. De forma opcional, se sugiere la utilización de alguna librería del estilo xUnit. Si bien existen varias librerías disponibles en lenguaje C++, se recomienda optar por CxxTest [10] o CppUnit [11]
6. Todo socket utilizado en este TP debe ser bloqueante (es el comportamiento por defecto).
7. El protocolo de comunicación tiene que ser binario (no texto) y no puede usarse una librería de terceros.

Referencias

[1] https://en.wikipedia.org/wiki/Wolfenstein_3D
[2] https://lodev.org/cgtutor/raycasting.html
[3] https://permadi.com/1996/05/ray-casting-tutorial-1/
[4] https://www.youtube.com/watch?v=vYgIKn7iDH8
[5] https://yaml.org/
[6] https://www.youtube.com/watch?v=VhctmeEebVo
[7] https://wolfenstein.fandom.com/wiki/Category:Wolfenstein_3D_sprites
[8] https://www.spriters-resource.com/snes/wolfenstein3d/
[9] https://www.spriters-resource.com/pc_computer/wolfenstein3d/
[10] https://cxxtest.com/
[11] https://en.wikipedia.org/wiki/CppUnit

2 División de Tareas

2.1 Germán Rotili

Modulo de Raycasting
Editor
Menú de Inicio
Configuración

2.2 Joaquin Lopez Saubidet

Modelo del juego
Logica de movimiento
Logica de ataque

2.3 Andres Visciglio

Comunicación Cliente-Servidor
Dinámica de partidas
Modulo IA
Aplicación de Threads

2.4 Luciano Ortiz

Renderizado de paredes, puertas, objetos, jugadores
Sonidos y musica
Display de HUD, armas, animaciones
Instalación

3 Evolución del Proyecto

3.1 Client

El cronograma propuesto es el mismo que se puede encontrar en la sección del enunciado, el el que se proponía un cronograma tentativo.

El cronograma real termino siendo distinto, mas que nada porque hubo muchas features de logica de juego que no se pudieron probar hasta tener una interfaz grafica funcionando. Además en el cronograma propuesto no se tomaron en cuenta tres semanas de desarrollo, que en el cronograma real si existieron, estas fueron las semanas del (07/01/2020) a la ultima de enero.

Semana 1: Ecuación de Raycasting. Inicio de las logicas de juego. Planteo de lógica de juego. Planteo de protocolo de comunicación.

Semana 2: Modulo de raycasting andando y dibujado de elementos "3D" sin texturas. Wrappers SDL.

Semana 3: Dibujado de texturas de las paredes. Inicio del Editor.

Semana 4: Finalización de la versión del editor sin guardado de mapas. Inicio de la configuración y guardado de mapas. Corrección de texturas.

Semana 5 Dibujado de objetos y enemigos. Configuración terminada. Implementación de animaciones.

Semana 6: Configuración integrada al cliente y el editor. Editor carga y guarda mapas. Objetos bloqueantes. Implementación de Features.

Semana 7: Puertas y paredes secretas. Sonidos y musicas. Mejora del editor y agregado de elementos del mapa, mejora de la selección de elementos y la ubicación de los mismos en el mapa. Implementación de Features.

Semana 8: Pantallas de login, game lobby y fin de partida. Implementación de Features.

Semana 9: Mejoras de UX, instalación e inicio de documentación. Testing y corrección de bugs. Implementación de Features.

Semana 10: Testing. Documentación. Armado del entregable.

3.2 Server

En el caso del server, las cosas no fueron implementadas por semanas sino que fue un avance general constaante. en el caso de la logica del juego, primero hubo un diseño de unas 3 semanas el cual fue variando en funcion de las distintas reuniones con el correcto. una vez diseñado se implementaron las clases principales como Map, Player, Position e Inventory. una vez echas, se paso al diseño de las armas y logica de disparo

3.3 Comunicacion

3.4 Modulo IA

Se considero que el modulo de IA debia desarrollarse una vez que se tuviera el juego en funcionamiento. De esta forma se podria saber que informacion mandar, que resolver y sobre todo, probarlo. Es por eso que se realizo en las ultimas semanas

4 Inconvenientes Encontrados

LUA Desde el lado del server, al inicio fue complicado entender el alcance esperado de este. en un principio entendimos que el modelo de juego seria compartido en el server y cliente. que el server se usaría de validación y el juego tendría que poder correr solo por su cuenta. esto retraso el inicio del diseño del server ya que se entendía que el diseño del juego requeriría tanto el diseño del servidor como el diseño del cliente para que este funcione de forma similar a juegos del

estilo modernos, siendo capaz de detectar colisiones y predecir el movimiento de jugadores entre updates del servidor. el modelo final utilizado es de mucha menor complejidad.

4.1 Comunicación

El modulo de comunicación fue cambiando drásticamente con el avance del trabajo practico. Varias necesidades demandaron una re-adaptación de la forma en la que se comunicaban el cliente y servidor. Dichas necesidades derivaron en la creación de distintas clases portadoras de información que son utilizadas tanto en el Cliente como en el Servidor, agregando una complejidad extra al manejo de la información. Sobre el final del trabajo, con la implementación de los Threads se tuvo que modificar de ambas partes el manejo de las variables para evitar inconsistencias, race-conditions o demás inconvenientes.

5 Análisis de Puntos Pendientes

Falta un refactoring general del código para emprolijar el proyecto en general. Por como esta implementado el server, seria simple agregar features que aumenten la calidad del juego como tal, por ejemplo. disminuir la velocidad cuanto te disparan, alguna tecla que disminuya la velocidad angular del jugador para apuntar mejor, cambio de velocidad dependiendo del arma. y asi se nos ocurrieron varias cosas que serian fáciles de implementar y agregarían mucho al factor jugabilidad.

Con respecto al modulo de IA, tiene mucho potencial para mejorar. Se podria aplicar raycasting en los enemigos para que determinen si los jugadores estan del otro lado de una pared. Tambien agregarles la opcion de buscar balas cuando se quedan sin y cambiar el arma a la que mayor daño tena.

Con respecto al cliente, por problemas de tiempo quedó pendiente implementar que reciba las armas disponibles del jugador para poder mostrar a qué armas puede o no cambiar, y tambien para comunicarle qué llaves tiene disponibles y poder dibujar en pantalla las disponibles

6 Herramientas

6.1 GitHub

Control de versiones
Organización del proyecto

6.2 SDL

Interfaz gráfica
Librerías de sonido

6.3 yaml-CPP

Archivos de configuración
Carga de mapas

6.4 Cmake

Compilación
Instalación

6.5 Valgrind

Debbuging
Checkeo de manejo de memoria

6.6 Latex

Documentación

6.7 LUA

El modulo de IA presento severos problemas a la hora de realizar la instalacion, lo cual atraso el testeo del mismo. Finalmente se encontro una opcion para hacerlo funcionar en los distintos ambientes de cada participante.

6.8 plantUML

Diagramas

7 Conclusiones

Al tratarse de un proyecto tan extenso con tantas partes que interactúan y se comunican entre si, resulto muy importante designar roles para poder ir desarrollando todas las partes en conjunto.

Encontramos que había partes del proyecto que eran difíciles de probar al no tener todavía una interfaz gráfica completa, capaz hubiese sido bueno armar prototipos gráficos para poder ir probando módulos de funcionalidad y lógica

de juego. Esto llevo a que una vez que pudimos realizar la interfaz de juego, tengamos que hacer un debugging general de a funcionalidad de juego en poco tiempo.

También notamos que al tratarse de un volumen de código muy grande, arrancábamos realizando código desprolijo para poder ir armando features, que luego, o dejaban dificultad en el futuro para agregar detalles y requerían un refactoring muy grande, que terminaban quedando como código desprolijo de todas formas. Sabemos que esto es un punto que se puede mejorar.

En cuanto a la situación del desarrollo a distancia, es distinto a lo normal, no conocer personalmente al resto del equipo y de todas formas lograr terminar un proyecto de esta magnitud nos demuestra a nosotros mismos que podemos adaptarnos a situaciones de índole extrema. Creemos que esto es una buena enseñanza para mundo laboral que seguramente vamos a vivir.

7.1 Conclusiones Particulares

7.1.1 Joaquin Lopez Saubidet

En este Tp me puse como Objetivo personal utilizar una forma de trabajo que inicie por un diseño del código a implementar, con un sistema de clases y secuencias detallado antes de programar absolutamente nada. Esta forma de trabajo me llevo a desarrollar código que considero fácil de expandir, simple para debuggear y legible. También me ayudo a pensar los problemas a resolver trabajando con un nivel mayor de abstracción e ignorando las particularidades del lenguaje y modelo. esto llevo a algunos problemas por falta de conocimiento del entorno; por ejemplo: un diseño que se basaba en eventos fue descartado porque los eventos que pueden ser implementados en c++ no servían de forma correcta. En resumen, esta regla auto impuesta para forzarme a trabajar con distintos niveles de abstracción y separando la parte de diseño de la parte de implementación me ayudo a poder ayudar a mis compañeros a elaborar modelos mas efectivos y principalmente a encontrar errores de diseño que estaban causando o podrían causar problemas.

7.1.2 Germán Rotili

Durante el proyecto supe administrar bien mis tiempos, lo que hizo que pueda ir separando bien las cosas que tenia que hacer e ir terminando de forma tranquila las distintas partes. Esto me dejo amplio tiempo para poder realizar correcciones y mejoras en UX, reworks de la configuración y agregado de elementos gráficos que creo que se pueden ver en el proyecto. En cuanto a aprendizajes personales, la buena administración de mis tiempos con el proyecto me mostró además que puedo ser de ayuda para resolver problemas que se podían encontrar el resto de mis compañeros cuando tenían que implementar cosas que requerían de partes que yo había programado, ya sea uso del editor, uso de las configuraciones o entender la secuencia de comunicación esperada durante los menús de inicio. Es bueno saber que puedo resolver mis problemas y ayudar al resto a trabajar los

suyos, y aceptar la ayuda del resto en los lugares que yo estaba faltante. Como algo mas particular, haber podido llegar a una visualización de elementos "3D" en un motor que gráfico que parece ser orientado hacia el desarrollo 2D es un logro personal interesante. Era requerimiento del proyecto, pero habernos sentado varias horas con Luciano y haber podido resolverlo, realizando prototipos y debuggeando muchísimo, sacamos una primera versión andando del raycaster en una semana y media, logro que antes de empezarlo pensábamos que nos iba a llevar mucho mas tiempo. Ahora en cuanto a dificultades personales con el proyecto, yo me tuve que pelear mucho con las interfaces de usuario, que son cosas que no parecen mucho una vez que están hechas pero que llevan mucho tiempo de probar y alterar detalles. En gran parte es porque me tuve que pelear con SDL, al no tener experiencia con otra librería gráfica, no se compararlo con nada, pero fue un desafío interesante.

7.1.3 Andres Tomas Visciglio

Como conclusion subjetiva del trabajo practico, considero que tuvo muy marcadas sus distintas etapas y que cada una presento dificultades muy diferentes. En primera instancia, tomo bastante tiempo definir un diseño correcto y global de como iba a ser el funcionamiento del servidor y el juego en general, lo cual no permitio avanzar con la programacion del mismo hasta entradas algunas semanas. Distinto sucedió con la logica de la comunicacion, que necesitaba de consensos tanto del cliente como del servidor, sobre formato, información, etc. Es por eso que el modulo obtuvo su forma final una vez que el servidor y el cliente estaban casi completos, con lo cual fue uno de los últimos módulos en implementarse. Mas aun, hasta ultimo momento recibió modificaciones por algunos features que necesitaban enviar distintas formas de información, lo cual podría haberse encarado de otra forma desde un principio. En relación a la distribución del trabajo, creo que fue acertada la división de tareas. El haber dividido el desarrollo de la lógica de juego de la comunicación permitió que trabajásemos de forma encapsulada, sin estar condicionando los avances de uno con otro. Esto nos permito tener ambos módulos funcionando casi en simultaneo y con una respuesta ante bugs muy efectiva, producto de la especialización de cada modulo.