Wolfenstein 3D
Manual de Proyecto

[75.42] Taller de programación
Segundo cuatrimestre de 2020
Repositorio en Github
Grupo 8

BERTOLOTTO, Francisco	fbertolotto@fi.uba.ar	102671
LÓPEZ NÚÑEZ, Agustín	alopezn@fi.uba.ar	101826
SANTONI, Mauro	msantoni@fi.uba.ar	102654
FERNÁNDEZ, Andrés	andyfer@fi.uba.ar	102220

Índice

1. División de tareas

- **Servidor:** Lógica general del modelo y del servidor. Implementación de IA en Lua. Implementación de protocolos y elementos necesarios para la comunicación. -> **Mauro Santoni y Francisco Bertolotto**.
- **Cliente:** Lógica general del cliente. Implementación de motor gráfico Ray Casting. Comunicación con el servidor. -> **Andrés Fernández**.
- **Editor:** Implementación completa del editor de mapas y el menú de configuración. -> **Agustín López Núñez**.

1.1. Mauro Santoni y Francisco Bertolotto

- Diseño e implementación de la lógica general del juego.
- Manejo integral del servidor en su totalidad.
- Implementación de protocolos de comunicación.
- Implementación de IA en Lua.
- API para la comunicación del juego con los Bots.
- Diseño de concurrencia en la ejecución de tareas.
- Comunicación con clientes.

1.2. Andrés Fernández

- Implementación del modelo de clientes.
- Implementación del motor gráfico Ray Casting.
- Implementación de la renderización de objetos en el mapa.
- Renderización de HUD y apartados de la interfaz de usuario.
- Renderización del audio del juego.
- Comunicación con el Servidor a través de threads.

1.3. Agustín López Núñez

- Diseño principal del editor utilizando Qt Designer.
- Implementación de comportamiento de todos los componentes pertinentes al editor (botones, grillas, menús, etc).
- Diseño del formato del mapa en formato YAML.
- Diseño del menu de conexión previo al juego y su comunicación con el server.

2. Evolución del proyecto

- **Semana 1:** Maquetación básica del editor en QT Designer. Diseño e implementación de carga de mapa a través de parseo con YAML. Ray Caster de paredes con texturas básicas. Parseo correcto del YAML.
- **Semana 2:** Aplicación básica de edición de mapas, con una grilla de botones que permiten agregar texturas al mapa. Lógica de movimiento y disparo. Procesamiento del disparo. Pickup de items. Agregado de configuraciones por archivo YAML.
- **Semana 3:** Creación de botonera de texturas en el editor que permiten funcionalidad point and click y drag and drop para carga de mapas. Implementación de threads sobre el servidor, finalización de lógica de juego con elementos que se mueven y/o desaparecen como puertas, RPG, paredes.
- **Semana 4:** Se agrega la posibilidad de scroll del mapa en el editor al ser grande la grilla y funcionalidad de point and click con click derecho. Implementación de IA en Lua. Sistema de comunicación cliente-servidor.
- **Semana 5:** Integración de cliente y servidor. Resolución de problemas de concurrencia en ambos planos. Mejoras sobre Ray Casting. Se permite resetear el cursor y mejoras sobre el editor. Mejoras en la estabilidad del servidor.
- **Semana 6:** Se comienza nuevo menú inicial en QT. Finalizacion de requisitos faltantes.
- **Semana 7:** Menú inicial completo, permitiendo configuración previa. Mejoras en la estabilidad y rendimiento del cliente y servidor. Correcion de bugs.

3. Inconvenientes encontrados

- Correcta instalación y posterior funcionamiento de la librería de parseo YAML. Esto derivó en la decisión de incluirla como parte del repositorio y paquete.
- Desarrollo de bot en Lua, principalmente por ser un lenguaje nuevo para los integrantes. Problemas de *race conditions* con el bot.
- Por el lado del Ray casting, su implementación fue compleja, especialmente la visualización de los objetos en un tamaño acorde. La falta de información sobre cómo dibujar sprites en este tipo de técnica, en comparación con la que hay sobre Raycasting, terminó volviendo a este apartado la parte más difícil del desarrollo del Cliente.
- Lograr estabilidad a lo largo de varios threads, ya que algunos mantienen sockets y otros lógica de juego.
- Data races a causa de las librerías gráficas que escapan a los integrantes, más específicamente debidas a QT y a SDL.
- Mejor manejo de rutas para facilitar la ejecución mediante instalación manual o mediante paquete.
- Generación de archivo de configuraciones para ser modificado en una carpeta sin permisos de root.
- Recibir cambios de alguien que ya no tiene mas vidas. Como existe una pequeña latencia entre lo que ve el cliente y lo que esta pasando, a veces ocurría que el cliente enviaba un disparo cuando en ese mismo instante moría; esto generaba problemas ya que estábamos procesando cambios de alguien "muerto".

4. Análisis de puntos pendientes

- Más tipos de bots disponibles para jugar.
- Más tipos de shortcuts para facilitar la creación de mapas.
- Posibilidad de soportar múltiples teclas presionadas al mismo tiempo.
- Renderización de sprites columna a columna (evita superposición en casos borde).
- Mayor variedad de sprites y texturas.
- Subir archivo deb a un repositorio PPA para su fácil acceso.
- Mejor uso de excepciones (en los tres módulos).
- Resize del mapa del editor hacia abajo del default.
- Mejoras visuales del editor.

5. Herramientas utilizadas

5.0.1. Control de Versión - Github

Se opto por Github por la familiriadad de todos los integrantes con la plataforma.

5.0.2. Compilación - G++

Para compilar el proyecto se utilizo g++ con estándar C++11.

5.0.3. Graficos

- **SDL2:** Se utilizaron librerías de SDL2 para todo lo referido a renderización de texturas y sprites (SDL_Image) durante el desarrollo del juego. También se usó para menús y pantallas con mensajes (SDL_TTF).
- **QT5:** Se utilizo principalmente para los menus principales y para el editor en su totalidad.

5.0.4. Audio

SDL2: Para la reproducción de música y sonidos durante el juego, se utilizó la librería SDL2 (SDL2_Mixer).

5.0.5. Debugging

- **CMake - Clion:** Para la fácil generación de ejecutables se utilizaron diversos CMake, esto junto con CLion permitió un fácil acceso al proceso de compilar y testear.
- **Netcat:** Se utilizo para pruebas simple de validación de envió y recepción de bytes.
- **Valgrind:** Se utilizo para verificar las perdidas de memoria.
- **Callgreen:** Se utilizo para analizar el comparta miento del programa, que funciones se llaman mas y cuales tardan mas.
- **Kcachegrind:** Se utilizo para observar graficamente los resultados obtenidos por Callgreen.
- **Tsan:** Se utilizo para buscar y eliminar posibles race condition en la ejecución del programa.

5.0.6. Google Cloud

Para probar la estabilidad en general se hosteo el servidor en una maquina provista por el servicio de Google Cloud. Esto permitió el correcto testeo de las funcionalidades online.

6. Conclusiones

6.1. Servidor

El manejo de múltiples threads de forma ordenada y evitando race conditions, fue sin duda un desafío importante, teniendo en cuenta que a los threads propios dedicados al funcionamiento del Servidor, se suman los threads de cada cliente, y todos comparten un único socket.

La integración del bot de LUA también fue un punto complicado, ya que es una librería relativamente complicada de utilizar de forma correcta, y a la cual había que adaptar a una lógica de IA que de por sí era bastante compleja.

Sentimos que la implementación de los mecanismos de comunicación con el cliente (incluyendo las colas bloqueante y no bloqueante), los Cambios enviados por el Servidor, los Eventos enviados por el Cliente, y la Network Connection que representa al socket, fueron ideadas de forma correcta y no tuvimos complicaciones relativas al funcionamiento de estas clases.

6.2. Cliente

Sin lugar a dudas, la parte más desafiante del trabajo práctico fue la renderización de objetos en pantalla de forma correcta. La técnica de Ray Casting tiene sus complicaciones, pero al existir una buena cantidad de información sobre el tema en Internet, esta parte del trabajo se volvió bastante más fácil.

En cambio, para dibujar los sprites en el suelo de forma correcta, sin que los objetos se muevan de forma extraña o se agranden o achiquen demasiado de acuerdo a la distancia, se volvió un trabajo mucho más difícil. Casi no hay material disponible sobre esto, y se intentaron formular muchas soluciones hasta hallar la correcta, lo que retrasó el trabajo práctico en general, ya que la correcta renderización de objetos es una herramienta de debug tanto del Cliente como del Servidor.

La comunicación con el Servidor fue formulada principalmente del lado del Servidor, y adaptada luego al Cliente. La reproducción de sonidos es relativamente trivial, aunque existen algunas race conditions cuyo origen es difícil de entender, ya que es propio de las librerías de SDL_Mixer.

El proyecto fue mucho más grandes de los que había realizado hasta el momento, fue difícil aplicar todos los conceptos de la POO y los nuevos conceptos aprendidos en la materia (por ejemplo, RAII) de forma ordenada y prolija.

6.3. Editor

Se obtuvo una mayor comprensión de la programación orientada a eventos debido a la necesidad de utilizar librerías, en este caso QT, orientadas a este paradigma. La utilización de la herramienta de QT designer aunque facilitó ciertas tareas no fué del todo útil para su uso con muchos Widgets (como es el caso del ConfigChecker).

6.4. General

Como conclusión general, nos pareció un trabajo práctico realmente difícil, a cuya fecha de entrega llegamos con los tiempos muy justos, incluso utilizando una parte del verano para avanzar con el mismo, posibilidad que sabemos que no existe en otros cuatrimestres. Sin duda, la dificultad en la correcta renderización de los objetos retrasó más de lo esperado el proyecto, lo cual conllevó demoras imprevistas.

Por otro lado, se consolidaron conocimientos sobre programación general, protocolos de comunicación, organización de proyectos y herramientas pertinentes. También se aprendió sobre el

desarrollo de un proyecto complejo en general, buenas prácticas y comunicación con otros programadores. Se utilizó mucho pair programming como herramienta de debug.