.UBAfiuba
FACULTAD DE INGENIERÍA

Taller de Programación I
(75.42/95.08)

Trabajo Práctico Final

Manual de Proyecto

Apellido y Nombre	**Padrón**
Inneo Veiga, Sebastian Bento	100998
Martinez Sastre, Gonzalo Gabriel	102321
Riedel, Nicolas Agustín	102130
Zbogar, Ezequiel	102216

Integrantes

Los integrantes del grupo 4 son:

- Inneo Veiga, Sebastian Bento, Padrón: 100998
- Martinez Sastre, Gonzalo Gabriel, Padrón: 102321
- Riedel, Nicolas Agustín, Padrón: 102130
- Zbogar, Ezequiel, Padrón: 102216

Enunciado

El enunciado consistía en recrear el juego Wolfenstein3D, creado por ID Software en 1992. En esta recreación había que implementar un sistema de juego multijugador online haciendo uso del protocolo TCP junto con un editor de niveles compatible para el juego. Además, se debía implementar el motor 3D usando la técnica de Ray Casting.

División de tareas

La siguiente división de tareas es *aproximada* y algunas partes fueron hechas por más de una persona, ya sea porque la empezó una y la siguió otra o fueron hechas a la par.

***Inneo Veiga, Sebastian Bento*:** Servidor.

- Lógica de los items, desde la creación hasta la interacción con el jugador.
- Parseo y armado de archivos de configuración y mapa en formato yaml.
- Socket.
- Lógica de colisión del jugador con paredes e items.
- Lógica de puertas.
- Mapa.
- Lógica de las acciones del jugador.

***Martinez Sastre, Gonzalo Gabriel*:** Cliente.

- Raycasting.
- Renderizado de texturas, tanto estáticas como dinámicas.
- Inclusión de sonido al juego.
- Comunicación del cliente con el servidor.
- Lógica de desplazamiento del jugador.
- Planteo inicial de colisiones del jugador con paredes.
- Lógica inicial de puertas y pasadizos.
- Visualización del leaderboard.
- Edición de texturas y sonidos para el juego.

***Riedel, Nicolas Agustín*:** Editor, Cliente .

- Editor de niveles (Totalidad)
    - Drag n Drop.
    - Scrolling.
    - Botoneras.
    - ToolBar.
    - Shortcuts de Teclado.
    - Exportar e importar archivos haciendo uso de Yaml-CPP.
    - Input de Texto.
    - Listas Seleccionables.
    - Animaciones Básicas.
    - Pop ups informativos.

    - Exportar e importar archivos haciendo uso de Yaml-CPP.
    - Integración de archivo .ui al proyecto.
- Cliente
    - Pantalla de Login.
    - Pantalla de opción, si se desea comenzar una partida nueva o unirse a una existente.
    - Pantalla de selección de mapa/partidas (Según corresponda), con listas seleccionables.
    - Uso de Stack de QWidgets para cambiar la visual de la ventana.
    - Cambios al flujo del cliente para adaptarlo a las ventanas.
    - Búsqueda de errores en tiempo de ejecución, prueba de distintas combinaciones de mapas.

Zbogar, Ezequiel**:** Servidor.

- Integración con Lua y lógica de la IA.
- Lógica de las múltiples partidas.
- Lógica de las Armas.
- Lógica de los Rockets.
- Lógica del Game Loop.
- Serialización de los mensajes enviados al cliente.
- Lógica de la comunicación con el cliente.
- Creación del Leaderboard.
- Lógica de la cámara de espectador.

Evolución del proyecto

A continuación se detallan los progresos realizados separados en semanas.

Progresos al 24/11

Cliente

- Raycasting básico (sin texturas) funcional. Sólo persona caminando en mapa vacío.

Editor

- Draft Básico.

Servidor

- Envío y recepción de datos en hilos distintos.

Progresos al 01/12

Cliente

- Raycasting con texturas (sin optimizar). Sólo persona caminando en mapa vacío.
- Clase contenedora de texturas y enum TextureID.
- Pantalla de logIn completamente funcional.

Editor

- Investigación en documentación de QT, ejemplos básicos de ventanas con ítems seleccionables.

Progresos al 08/12

Cliente

- Raycasting fluido y con shading. Sólo persona caminando en mapa vacío.
- Conexión con servidor: recepción de coordenadas y envío de instrucciones de movimiento (se manda una instrucción cuando se presiona la tecla y otra cuando se la suelta).

Server

- Primera versión del Gameloop (Lee una instrucción por cliente si es que las hay, cambia el estado de movimiento de los jugadores y actualiza sus posiciones según corresponda).
- Primera versión del protocolo de comunicación, donde a cada jugador se le envía su actual posición y la del resto de los jugadores.
- Draft de la lógica de disparo.

Editor

- Uso de Qt Editor, integracion de archivo .ui al proyecto.
- Ejemplos de prueba Drags, mostrar imágenes, etc.

Progresos al 15/12

Server

- Draft del modelo de los ítems.
- Lógica de disparo.

Cliente

- Raycasting fluido con inclusión de texturas y jugadores con rotación.

Editor

- Integración de ToolBar en editor precario
- Lista de items imprimibles y seleccionables.
- Mapa imprimible, pero incapaz de recibir drops.

Progresos al 22/12

Server

- Modelo de los ítems obtenibles (recuperadores de vida, armas y tesoros).
- Ya se puede inicializar los jugadores y el mapa directamente del archivo config.yaml.
- Se implementa cola bloqueante para el “send”.
- Lógica de disparos terminada.
- Módulo de Lua.

Cliente

- Se separó su funcionamiento en 3 hilos diferentes y se solucionaron los problemas de performance al conectarse al servidor.
- Se comenzó con el diseño de la interfaz del jugador.

Editor

- Tiempo dedicado a leer documentación y ver ejemplos en la misma.
- Se decide empezar el Editor de 0.

Progresos al 2/2

Server

- Lógica de colisión.
- Mejora en la clase Configuration.
- Más parámetros modificables del archivo config.yaml.
- Finalizada la integración con Lua
- Sistema de múltiples partidas (Básico).

Cliente

- Interfaz básica del jugador con imagen de fondo y HUD inicial.
- Mejora de performance de raycasting.
- Desarrollo inicial de lógica de puertas y renderizado.

Editor

- Lista seleccionable con items arrastrables.

- Mapa recibe Drops, pero problema al calcular ubicación.

Progresos al 9/2

Server

- Selección de múltiples mapas al crear partida.

Cliente

- Interacción inicial del cliente con el servidor.
- Renderizado de texturas dinámicas.
- Recepción del mapa.

Editor

- Drag n Drop funcional,
- se agrega clase TrashBin.

Progresos al 16/2

Server

- Comienzo de implementación de las puertas.
- Implementación del ítem llave.
- Modificaciones en updatePlayer.
- Caída de items al morir.
- Sistema de múltiples partidas multi-threading, (Antes se podía atender un solo cliente a la vez a la hora de crear/unirse a una partida).

Cliente

- Incorporación de sonido.
- HUD y armas del usuario finalizadas.

Editor

- Ajustes de diseño.
- Ajuste de performance (Se procede a tener una sola instancia de cada Pixmap imprimible).

Progresos al 23/2

Server

- Agregado envío de sonidos.
- Implementación de Rocket Launcher y Rocket.
- Implementación final de la lógica de las puertas.
- Implementación de la llave.
- Correcciones al quedarse sin balas el jugador.
- Corregido el envío de texturas del jugador en base al arma que tiene equipada.
- Envío de sonidos que hacen otros jugadores

Cliente

- Renderizado de jugadores caminando.

- Finalización de renderizado puertas y pasadizos.
- Elaboración de visualización de leaderboard.
- Detalles finales de la interfaz del usuario.
- Rehace pantalla de LogIn.
- Pantalla de selección de partida.
- Integración final del cliente con QT.

Editor

- Exportación / Importación de archivos en YAML.
- Editor Completamente Funcional.
- Pop ups para notificar errores y ventana de ayuda.

Inconvenientes encontrados

Server

- En la implementación de los ítems se decidió hacer uso del patrón double dispatch. El problema es que para ello 2 clases debían conocerse entre sí y esto llevaba a problemas de doble inclusión. Se resolvió haciendo uso de forward declaration de una de las clases y la inclusión del .h en el .cpp de la otra clase.

Cliente

- En la implementación de la pantalla para seleccionar partida era necesario ir variando entre distintos widgets visibles, la solución a esto fue la clase **QStackedWidget** .
- La implementación del renderizado de las puertas fue realmente complicada, puesto que para poder realizar dicha tarea era necesario conocer el punto exacto de impacto del rayo sobre la superficie, lo cual era imposible de saber con la implementación de ese momento del raycasting. Afortunadamente, tras varias operaciones matemáticas se logró realizar la tarea.
- La documentación de SDL no era tan informativa como se esperaría.

Editor

- Aprender Qt fue mucho más complejo de lo esperado, se dedicó más horas a leer documentación que a escribir código.
- La documentación de Yaml-Cpp (La única biblioteca para exportar YAML)que encontramos para c++) tiene nula documentación.

Análisis de puntos pendientes

Si bien el resultado final es completamente jugable y respeta la consigna del trabajo, varios detalles ideados por el grupo no pudieron ser llevados a cabo, principalmente, por cuestiones de tiempo. A continuación, se enumeran algunos de ellos.

- Animaciones de misil de rocket launcher explotando, y de jugadores muriendo y disparando. Las texturas se encuentran en el repositorio pero no lograron ser incluidas en el proyecto.
- Incorporación de minimapa visualizable durante el juego. Si bien este fue realizado con fines de depuración del raycasting en las primeras semanas, fue dejado de lado y no pudo ver la luz en el producto final.
- Se pueden notar algunas inconsistencias en cuanto al envío y reproducción de algunos sonidos.
- El leaderboard en algunos casos puede renderizarse sin la imagen de fondo.

Herramientas

Las herramientas utilizadas para llevar a cabo este trabajo práctico fueron:

Git/Github
Para control de versiones y alojamiento del trabajo.

GCC
Compilador de los archivos cpp.

CMake
Herramienta para automatizar la compilación del proyecto que crea un Makefile para ello.

Make
Junto con CMake ayudaron a automatizar la compilación del trabajo.

Valgrind
Usado para buscar leaks de memoria y otros errores tanto en el servidor como en el cliente.

GDB
Junto con valgrind fue usado para buscar errores que no detectaba.

QtCreator
Usado en un principio para el desarrollo del código en Qt.

Qt Designer
Usado para generar los archivos “.ui”.

Google Docs
Para escribir los informes.

Trello
Para mantener un seguimiento de qué características del trabajo nos faltaba hacer o estaban en desarrollo.

CLion/Sublime text
Para el desarrollo del ćodigo.

Discord
Para comunicarnos y tener sentadas cuestiones menos formales del proyecto.

Conclusiones

Tras meses de desarrollo, el equipo se encontró con una gran serie de desafíos interesantes que pudieron ser analizados en profundidad y resueltos gracias a los conocimientos adquiridos en clase. Además, la experiencia de llevar a cabo un proyecto más grande de lo habitual de principio a fin, resultó en una experiencia entretenida y, a su vez, enriquecedora para todos los integrantes del grupo.

Honestamente, si bien es verdad que existen varios detalles que podrían ser pulidos y features que podrían ser agregados, el equipo se encuentra complacido con los resultados obtenidos y, asimismo, satisfecho por haber logrado cumplir la consigna asignada en su totalidad.

El trabajo en equipo fue pulido progresivamente y se convirtió en el núcleo fundamental de la actividad. Las buenas vías de comunicación implementadas contribuyeron a que no surgieran demasiados conflictos durante el desarrollo.

En conclusión, se puede afirmar que el presente trabajo fue una experiencia muy fructífera en términos generales al permitir a cada integrante usar nuevas herramientas y técnicas de programación.