Counter Strike 2D

Manual de proyecto

Taller de programación - 1° cuatrimestre 2025 - Cátedra: Veiga

Federico Honda - 110766

Martina Gualdi - 110513

Santiago Varga - 110561

Thiago Baez - 110703

.UBAfiuba
FACULTAD DE INGENIERÍA

Introducción

En este informe se presentarán las tareas realizadas por cada integrante, las herramientas utilizadas y se incluyen reflexiones finales, desafíos, alcances y conclusiones.

Desarrollo de tareas

Federico Honda: trabajo completo en el servidor, hilos para clientes, validaciones de partida y envío de estados de la misma a los clientes.

- **Conexión del lado del servidor**: Manejar la entrada de nuevos clientes que desean jugar al juego, teniendo en cuenta posibles errores que puedan suceder en el proceso y manejarlos de una manera *friendly* para el usuario.
- **Definición de protocolo y snapshot**: Qué información será enviada, en que orden y cómo, mediante el uso de sockets, para que luego del cliente se replique la lógica y se pueda recibir todo sin perder información en el proceso.
- **Uso de Queues del servidor:** Definir cómo se utilizarán las colas, quienes serán los dueños de ellas y en qué momento se debe encolar y desencolar de las mismas en la lógica del servidor para el manejo de las partidas.
- **Manejo de hilos para el cliente en el servidor**: Cada nuevo cliente tendrá sus propios hilo "Sender" y "Receiver" del lado del servidor, los cuales se encargaran de recibir los comandos que envía el mismo, encolarlos, y desencolar snapshots para que pueda renderizar el juego en el estado actual por pantalla.
- **Manejo de hilo para la partida:** Se tiene un hilo "Gameloop" el cual se encarga del manejo completo del flujo de la partida; actualizando las posiciones y estados de los jugadores, realizando validaciones sobre las acciones (por movimiento o por tiempo) de los mismos y controlando cuando la partida debe terminar, haciendo aviso a todos de cada cambio (broadcast). Algunas de ellas son:
  - Movimientos.
  - Disparos.
  - Dropeo de un arma.
  - Levantar un arma.
  - Cambio entre las armas disponibles.
  - Zonas y tiempos para realizar compras.
  - Colisiones (más en "Físicas del juego")
  - Espera de los jugadores a que se complete la sala.
  - Casos de desconexión en medio de partida.
- **Múltiples partidas:** Permitir que múltiples clientes puedan unirse y crear nuevas partidas para ellos y los usuarios que se quieran unir, pudiéndose dar todas en simultáneo. Para esto se dio uso

a un Monitor de Partidas.

- **Valores configurables:** Que se tenga acceso a un archivo "configuracion.yaml" el cual te permite modificar cualquier valor significativo de las partidas (vida, dinero, cantidad de municiones a un arma específica, ángulo de visión, etc), permitiendo que el balanceo de las mismas sea mucho más fácil ya que no se necesita volver a compilar luego de modificar el mismo.
- **Físicas del juego:** Validar desde los movimientos de los jugadores contra las paredes hasta si una bala impactó contra una de ellas o contra un jugador, haciendo daño.
- **Entidad Mapa:** Poder leer del archivo ".yaml" y cargar los datos que el servidor va a necesitar para realizar validaciones y enviar a los clientes.
- **Cierre ordenado:** Al terminar una partida, se eliminan a los jugadores (clientes) de la misma de forma que ningún proceso se quede "esperando" a que alguna loop termine o que se bloquee por un socket.
- **Sin leaks de memoria:** Que al terminar una partida, sea cual sea el motivo, no se pierda memoria en ningún sector del código a la hora de cerrar el server.

Martina Gualdi: llevó a cabo el desarrollo del editor de niveles gráfico, lobby, instalador y tests.

- **Editor de niveles:** sus features principales son, por ejemplo, drag&drop para elementos del mapa, pincel para pintar el piso, click para setear el fondo del mapa. También se cuenta con un marcador de zonas para seleccionar el lugar en el que se quiere posicionar spawns y plantación de bombas. Los mapas creados también se pueden editar una vez guardados
- **Lobby:** el lobby cuenta con dos etapas principales, la conexión con el server y la creación/unión a partidas. En la primer etapa se debe ingresar host, port y username. En la segunda etapa se podrá crear una partida (lo cual permitirá elegir el mapa en el cual jugar, que puede ser uno creado por el mismo usuario en el mismo editor de niveles), listar las partidas disponibles y seleccionar una de estas para unirte (siempre y cuando la misma no esté ya iniciada).
- **Instalador:** el instalador ejecuta la compilación, tests e instalación de dependencias necesarias para la ejecución del programa. Una vez corrido el comando correspondiente, se tendrán disponibles 3 iconos en el escritorio para ejecutar tanto el editor como el juego en sí (server y cliente).
- **Tests:** buscan probar la funcionalidad de los protocolos, principalmente en envío y recepción del snapshot.

Santiago Varga: desarrollo de las armas, lógica de bomba dentro de la partida y actualización de su estado, logica de mercado.

- **Herencia en Armas:** para las armas se eligio adoptar un sistema de doble herencia ya que todas estas (*Cuchillo, Bomba, Glock, Ak-47, Awp, M3*) tienen ciertas cualidades que las asemejan pero también diferencias en su implementación. De este modo tenemos una primera clase abstracta *Arma*, de la cual derivan *Bomba*, *Cuchillo* y otra clase abstracta *ArmaDeFuego*, derivando de esta las demás.
- **Sistema de precisión, daño y cadencia:** cada *ArmaDeFuego* tiene diferente precisión dependiendo de una probabilidad dada por una distribución normal uniforme, haciendo que no siempre se pueda atinar al objetivo. También, otro aspecto que afecta al daño de las armas es la distancia,

mientras mas lejos se encuentren los jugadores, mas se reducirá el daño hasta que se llegue al máximo alcance del arma. Si se sobrepasa ese alcance el arma no hara ningun daño. Por ultimo, cada arma tiene una cadencia específica que modifica la velocidad con la que esta se puede disparar. Haciendo que armas más pesadas, como una escopeta o un francotirador, tengan mas tiempo de espera para disparar y equilibrando el juego.

Un caso excepcional es el caso de la *Awp*, que siempre hace el máximo daño, y el *Cuchillo*, que realiza un *random* del minimo al maximo daño posible para cada puñalada, pero no reduce el daño con la distancia.

- **Sistema de perdigones (*M3*):** al ser una escopeta, la *M3* tiene un sistema de perdigones (3 elementos *Munición*) los cuales se disparan en forma de cono. Estos son completamente independientes entre si y hacen daño cada uno por su cuenta. Otro particularidad de esta arma es que la forma de calcular su daño es diferente al de la demás armas. En este caso, al ser un arma de muy corta distancia, su daño debe reducirse mucho más que en los otros casos, haciendo que sea necesario un factor arbitrario y una fórmula diferente que genere una funcion con decaimiento mas rapida.
- **Sistema de rafagas (*Ak-47*):** esta arma es una ametralladora de rafagas, por lo tanto hubo que diseñar un sistema para controlar que cada vez que se ejecute un disparo se puedan visualizar tres balas (elementos *Munición*). Para esto se estableció un *chrono* el cual controlaba el estado de la rafaga que actualmente se estaba disparando, dejando tiempo entre cada bala para poderlas apreciar dentro de la partida. También son independientes entre si como en el caso anterior gracias a que son elementos de la clase *Munición*.
- **Lógica de *Bomba*:** la *Bomba* es un aspecto central en el desarrollo de las rondas del juego y esta cumple con diversas características únicas:
  - Existe una sola por partida.
  - Se asigna aleatoriamente a un jugador *TT* en cada ronda.
  - Tiene los siguientes estados: *SIN PLANTAR, PLANTADA, DESACTIVADA, DETONADA*.
  - Se puede levantar y dropear como un arma normal.
  - Solo puede ser manipulada por jugadores del equipo *TT*.
  - Puede ser plantada en las zonas asignadas (puede ser una zona o dos, dependiendo del mapa).
  - Una vez plantada en zona, puede ser desactivada por un jugador del equipo *CT*.
  - Si no es desactivada, la bomba explotara luego de un tiempo, terminando en una victoria para los *TT*.
  - Al explotar, cumplira con su papel como arma, haciendo daño en un area bastante considerable y reduciendo este con la distancia. El daño solo se aplica al equipo *CT*.
  - Al ser desactivada, la ronda terminara automáticamente con una victoria para los *CT*.
  - Si la *Bomba* se encuentra plantada y todos lo *TT* mueren, la partida continuara su curso hasta que se accione sobre esta (detonacion por tiempo o desactivación).
- **Lógica de Mercado:** los jugadores pueden comprar las armas principales (*Awp, Ak-47, M3*) solo cuando se encuentran dentro de la zona y tiempo de compra al inicio de cada ronda. Asimismo sucede para las balas, pudiendo comprar en este caso tanto balas primarias como secundarias (*Glock*). Cada arma tiene un máximo de compra de balas, limitado por la cantidad total que esta puede tener.

Thiago Baez : trabajo completo del lado del cliente, sin contar el Lobby.

- **Dibujador:** desarrollo de toda la interfaz gráfica del juego. Para esto se utilizó SDL2, más especificamente el wrapper hecho para C++, SDL2pp. Fue un gran desafío aprender sobre SDL2 y toda la parte gráfica. La sección gráfica del juego cuenta con algunas cosas interesantes:
  - HUD: muestra la información vital del jugador en tiempo real, sin interrumpir la acción del mismo. En este podemos encontrar info como la salud o vida restante del jugador, su saldo actual, el tiempo restante de la partida, la cantidad de balas que tiene el arma actual, si este posee o no la bomba o si se encuentra en tiempo y forma para realizar una compra.
  - Jugador: se muestra al cliente principal centrado en la pantalla, siendo el mapa quien pareciera moverse lugar (efecto de cámara) junto con el resto de los jugadores. El jugador cuenta con la posibilidad de elegir diferentes skins al inicio de la partida. También se incluyen animaciones mientras camina, dispara o se muere.
  - **FOV (Field of vision):** en español, campo de visión, esta es una forma configurable, de que el cliente se le oscurezca el resto de la pantalla que no está viendo directamente. Esto fue un punto bastante difícil de hacer.
  - Armas y balas volando.
  - Diferentes mensajes emergentes en pantalla, como “Los Terrorists han ganado!”.
  - **Cuadro de estadísticas:** se muestran las estadísticas básicas de la partida, puntaje de cada equipo, ronda actual, estado actual de cada jugador, sus puntos y sus muertes.
  - **Tienda:** ventana de compra de armas y balas para el jugador.
  - Animación de la explosión de la bomba
- **Sonido:** implementación del sistema de sonido del cliente. Se utilizó SDL2pp Mixer, pudiendo reproducir múltiples sonidos de forma simultánea con control de volumen dinámico. El objetivo fue sincronizar los eventos del juego con sonidos específicos, y adaptando el volumen según la posición relativa del jugador (si se va lejos de donde esta ocurriendo un evento no lo escucha). Esta sección del proyecto implicó conocer cómo funciona la mezcla y reproducción de audio con SDL2:
  - Cap de sonidos (máximo 4 a la vez)
  - Sonidos simultáneos limitados: para evitar saturación auditiva, se implementó un sistema de conteo que limita la cantidad de sonidos iguales activos al mismo tiempo.
  - Pasos
  - Disparos: dependiendo del arma y cercanía.
  - Compra de armas y balas.
  - Explosión de la bomba y pitidos de cuenta regresiva con frecuencia creciente a medida que se acerca la detonación.
  - Ganador de ronda: se reproduce la voz indicando el equipo que ganó y el motivo (detonación, desactivación).
- **Event Handler:** interpreta todos los eventos del usuario en el cliente. Se conecta con SDL2 para leer entradas del teclado y mouse, y las traduce en comandos que luego son enviados al servidor. Principales funcionalidades:
  - Movimiento del jugador
  - Rotación y disparo

    - Interacción con el mercado
    - Plantar o desactivar bomba
    - Cambiar de arma
    - Tirar arma o levantar una del piso
    - Ver estadísticas (TAB)
    - Sistema de desconexión con doble confirmación (Q y ENTER)
- **Client Map:** carga y procesa el mapa recibido a través del servidor en formato YAML. A partir de esa información, crea una estructura interna con las distintas texturas y sus prioridades de renderizado:
    - Renderizado ordenado: cada elemento se clasifica por una prioridad que define el orden en que deben dibujarse (de atrás hacia adelante).
    - Cacheo de texturas: para optimizar el rendimiento y la memoria RAM, se implementó un cache de texturas ya cargadas y evita leer imágenes repetidas desde disco.
- Implementación de hilos sender y receiver para interactuar con el servidor a través del protocolo
- YAML de la configuración de las dimensiones de la pantalla y FPS del cliente.
- **Constant Rate Loop:** Evita que el juego corra más rápido o más lento dependiendo del rendimiento del hardware y sincroniza la lógica del juego y el dibujado a una velocidad estable.
- Otras clases como Parser de Spritesheets

Herramientas utilizadas

El proyecto se ha desarrollado en C++ bajo POSIX 2008, utilizando herramientas y tecnologías como:

- **Sistema Operativo:** Distribuciones Ubuntu 24.04 o Xubuntu 24.04.
- **Compilador:** GNU Compiler Collection (GCC) v2.
- **Lenguaje de Programación:** C++20.
- **Entorno de Desarrollo Integrado (IDE):** Visual Studio Code.
- **Bibliotecas Gráficas:** SDL2, SDL2pp y Qt6.
- **Framework de Pruebas:** GTest.
- **Herramienta de Chequeo de Memoria:** Valgrind.
- **Formato de Configuración:** YAML.
- **Control de Versiones:** GitHub.
- **Editor utilizado:** VSCode en todos los casos.

.UBAfiuba
FACULTAD DE INGENIERÍA

Reflexiones

Desafíos

Uno de los principales desafíos del proyecto fue el aprendizaje del uso de bibliotecas gráficas, tanto SDL para el juego como Qt para el lobby y editor de niveles.

Alcances

El proyecto logró alcanzar los objetivos estipulados por la cátedra, incluyendo:

- Lobby de conexión y recepción de jugadores.
- Cliente multithreaded para el envío y recepción de mensajes cliente-servidor.
- Servidor multithreaded con el mismo objetivo, que además se encarga del manejo del ciclo de juego.
- Aprendizaje de la librería SDL2, con todo lo que esto conlleva (entender de pixeles, dimensiones, renderizado, FPS, etc.)
- Creación de un protocolo binario que permite la correcta comunicación entre cliente-servidor.
- Utilización de polimorfismo para el desarrollo de las diferentes armas involucradas en el juego.
- Editor de niveles gráfico encargado de permitir el diseño que el usuario desee de un mapa y la creación de un YAML con la información necesaria para luego ser utilizado en el juego.

Sugerencias para próximos cuatrimestres

Consideramos que puede ser de suma utilidad algunas clases más de explicación del uso de herramientas gráficas para todo el desarrollo de la interfaz con el usuario.

Conclusiones

Desde el equipo nos encontramos satisfechos con el desempeño y resultado del proyecto.

Consideramos que logramos alcanzar los objetivos en su totalidad en tiempo y forma con una buena organización y comunicación del equipo.