Manual de Proyecto

Participación del equipo

Manuel se encargó de todo el desarrollo de la interfaz del cliente. Implementó el menú principal, pantalla de lobby y demás pantallas del menú, y toda la visualización del juego en tiempo real, tomando la información de StateGame y representando gráficamente. Desarrolló la lógica de renderizado, el manejo de eventos del usuario, y la integración con SDL y Qt. También participó activamente en el desarrollo del editor y ayudó con el planteo inicial de algunas mecánicas del juego y experiencia de usuario.

Nicolas se encargó del desarrollo completo del servidor, diseñó e implementó el protocolo de comunicación, escribió pruebas para garantizar el correcto funcionamiento del sistema y actualizó el instalador para que resuelva dependencias, compile el proyecto y organice los archivos en rutas estándar del sistema para su ejecución mediante rutas absolutas. Además, se encargó de realizar la documentación pedida. También se encargó de la estructura del map/mapdata y la lectura de yaml.

Marina se encargó de la totalidad del desarrollo de la lógica del juego, de las clases Game, Team, Player, Hitbox, GameMap y Store. Implementó el cambio de fases, físicas, colisiones, movimiento, disparo, plantado y desactivado de bomba, el sistema de compra de armas y munición, además del envío del estado del juego mediante StateGame y la ejecución de acciones del jugador. Por último, participó en la redacción de la documentación pedida.

Milagros se encargó del desarrollo del editor del juego y colaboró en tareas de refactorización del cliente. Implementó la interfaz y la lógica necesarias para crear mapas con el editor, permitiendo seleccionar e insertar distintos elementos en cada posición del terreno, así como guardar el resultado en un archivo.

Desarrollo semanal

Comenzamos el proyecto desarrollando una versión básica del juego, con el objetivo de validar la comunicación cliente-servidor, el renderizado visual y la lógica principal. Esta versión inicial, aunque sólo permitía el movimiento de los jugadores, nos permitió asegurarnos de que todos los componentes fundamentales funcionaban correctamente.

Paralelamente, durante la primera semana se planteó una arquitectura funcional para el servidor. Posteriormente, se llevó a cabo una refactorización para optimizar dicha arquitectura dado que ésta era demasiado ineficiente. Desde entonces, el trabajo en el servidor se enfocó principalmente en corregir problemas menores y optimizar detalles.

Una vez que logramos tener esta base estable, fuimos agregando progresivamente funcionalidades más complejas. Entre ellas: disparo de balas y detección de colisiones, interacción con objetos del

mapa (como la bomba), lógica de equipos, rondas y condiciones de victoria, tanto en la lógica del juego como en su representación visual, integrándose también en el servidor y el protocolo de comunicación.

A medida que avanzábamos, también fuimos corrigiendo errores y refactorizando partes del código para mejorar su estructura y rendimiento. Esta estrategia nos permitió mantener una base estable durante todo el desarrollo y facilitar la integración de nuevos módulos sin romper lo anterior.

Herramientas utilizadas

IDE: Visual Studio Code.
Control de versiones: Git y GitHub.
Linters y formateadores: clang-format, cpplint, cppcheck.
Compilado: Makefile, CMake
Testing: GDB, Valgrind
Otorgado por la cátedra: Socket, Thread
Otros: Bash, Docker.

Documentación consultada

Material de la cátedra, ejemplos provistos por la materia (cliente/servidor, uso de sockets y threads).
Documentación oficial de SDL y Qt

Sugerencias para la cátedra

Hemos notado que los enunciados de los trabajos prácticos a veces resultan poco específicos, lo que genera confusión respecto a qué aspectos se espera que prioricemos o consideremos en la resolución.

Sugerimos que, en futuras consignas, se brinde una redacción más clara y detallada, incluyendo ejemplos o criterios de evaluación cuando sea posible, para facilitar la correcta interpretación de los objetivos del ejercicio.

Por otro lado, proponemos incorporar una mayor diversidad en las temáticas de los proyectos. Si bien el desarrollo de juegos es una experiencia interesante, creemos que también sería enriquecedor contar con opciones orientadas a otros dominios. Esto permitiría a los estudiantes abordar problemáticas más variadas, posiblemente más alineadas con sus intereses personales o con los perfiles profesionales que buscan desarrollar.

Sería interesante que se ofrecieran distintas propuestas de trabajo práctico entre las cuales los equipos puedan elegir, permitiendo así una experiencia más personalizada sin perder de vista los objetivos formativos de la materia.

Dificultades encontradas

- Comenzamos planteando que cada cliente tenga su propia instancia de game para simular y ver de manera fluida antes de recibir los mensajes del servidor, pero nos dimos cuenta que no era conveniente y lo modificamos a una única instancia de game en el servidor.
- Inicialmente, el movimiento de los jugadores se manejaba mediante una llamada a execute() en cada iteración del gameLoop. Esta lógica fue mejorada para que execute() se invoque sólo al comenzar un movimiento o al cambiar el vector de velocidad, y que el estado se actualice en función del deltaTime con update(), evitando que el servidor reciba un exceso de acciones innecesarias.
- La integración con Qt y SDL fue complicada debido a no solo errores de compatibilidad en algunos casos (por ejemplo, SDL crashea si es inicializado después de Qt), que las librerías usa sino también que las librerías usan métodos muy distintos para la captura de eventos del usuario.
- Uno de los principales desafíos fue plantear una arquitectura adecuada para el servidor. Inicialmente, no implementamos hilos separados para *Receiver* y *Sender*, lo que nos llevó a utilizar un esquema de *busy wait* que consumía muchos recursos del CPU. Posteriormente, al reestructurar la arquitectura, surgieron complicaciones relacionadas con el cierre controlado de estos hilos y su correcta comunicación con las instancias de *Menu* y *Match*.
- Se presentaron dificultades al intentar evitar operaciones bloqueantes en el hilo *Receiver*, particularmente en el momento de recibir el nombre del cliente. Decidimos que esa responsabilidad no debía recaer sobre *Receiver* para mantener su ciclo limpio, pero conectar correctamente toda la lógica necesaria —desde la recepción hasta la configuración del nombre y el arranque de los hilos asociados— resultó engorroso y derivó en un código que quedó poco elegante.

Estado final

Pudimos llegar con todas las funcionalidades requeridas, aunque con algunos puntos que podrían mejorarse:

- El editor está incompleto y es poco intuitivo (quizás mejore hasta la entrega).
- Actualmente, el servidor envía el estado completo del juego en cada actualización; una posible mejora sería implementar un sistema de envío de cambios para optimizar la comunicación.
- El código podría beneficiarse de una mayor simplificación y modularización.
- Implementar las configuraciones desde el cliente.
- El protocolo podría generalizarse mediante la incorporación de un método *send_message*, unificando el envío de mensajes similar a como están las responses.

Cambios que haríamos si volviéramos a empezar

Si tuviéramos la oportunidad de comenzar nuevamente el proyecto, hay varios aspectos que mejoraríamos:

- **Diseño temprano de un protocolo más apropiado**: Comenzar desde el inicio con un protocolo de comunicación bien estructurado y flexible habría facilitado la integración entre cliente y servidor, reduciendo la necesidad de refactorizaciones posteriores.
- **Sincronización entre áreas**: En el desarrollo inicial hubo una descoordinación entre los responsables del servidor y del cliente. Por ejemplo, mientras se avanzaba con la arquitectura del servidor, el cliente aún no estaba disponible para realizar pruebas reales, ya que el foco estaba puesto en el desarrollo gráfico. Esto dificulto la validación de la lógica del servidor y generó demoras evitables.
- **Mayor integración entre los miembros del equipo**: Aunque es lógico que cada integrante se enfoque en un área específica, creemos que una mayor participación cruzada en las distintas partes del sistema (por ejemplo, entender mínimamente cómo estaba estructurado el cliente o el servidor) habría mejorado la comunicación, la comprensión del sistema en su conjunto y la colaboración general.