Programación 2

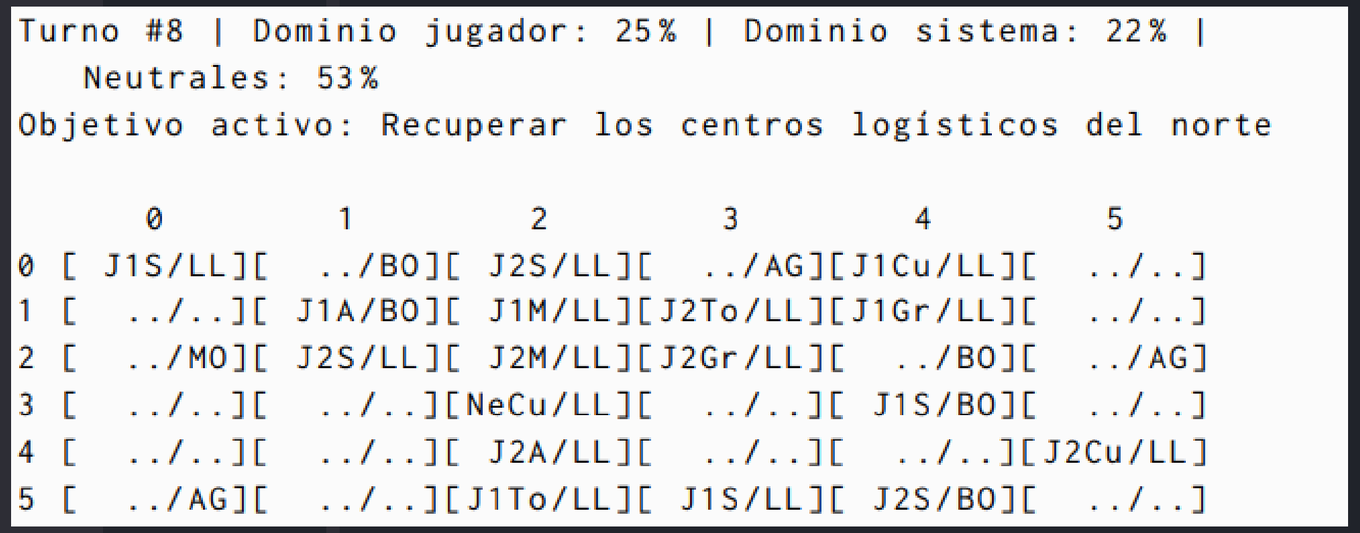
Proyecto Final
“Rival Frontiers”

<<Por="Cárdenas Soto, Tadeo Joaquín”;
<<”Cárdenas Carreras, Dahlia”;
<<”Rodenas Rocha, Oscar”;
<<”Yañez Encalada, Aldrian”;

Índice

¿De qué trata el proyecto?

El proyecto trata de desarrollar Rival Frontiers, un juego de estrategia por turnos en consola donde el jugador controla una civilización que debe enfrentarse a guardianes controlados automáticamente por el sistema. La acción ocurre en un mapa 2D con distintos terrenos, unidades y edificios que influyen en cada movimiento. El objetivo es expandir el dominio, gestionar recursos y completar misiones antes de que el sistema recupere el control del mapa.



}

¿Qué arquitectura utilizamos para el proyecto?{

Include

Esta carpeta contiene todas las cabeceras (.h)

src

Esta carpeta contiene los archivos fuente (.cpp)

El proyecto Rival Frontiers se construyó siguiendo un modelo de Arquitectura Modular.

}

Explicación de cabeceras(.h)

Unidad.h

El archivo Unidad.h define toda la estructura fundamental que usan las unidades dentro del juego.
Dentro del mismo archivo también se declaran las clases derivadas: Soldado, Arquero, Caballero, Mago e Ingeniero, cada una heredando de Unidad.

Mapa.h

Mapa.h define la estructura fundamental que representa el escenario principal del juego. En esta clase se administra el mapa completo como una matriz de objetos Celda, cada una conteniendo información del terreno, las unidades, edificios u otros elementos presentes.

Terreno.h

El archivo Terreno.h define toda la estructura responsable de modelar los distintos tipos de terrenos del mapa en el juego. Esta parte del sistema es esencial porque determina cómo afecta el entorno al movimiento, defensa y estrategia de las unidades.

Recursos.h

Define la estructura encargada de manejar los diferentes recursos que intervienen en la economía del juego. Aquí se encuentra la clase Recursos, que representa la cantidad de comida, metal y energía disponibles para un jugador o un edificio. Estos recursos juegan un papel fundamental, ya que determinan si una unidad puede activar una habilidad, si un edificio puede ser construido, o si una acción estratégica puede llevarse a cabo durante un turno.

Celda.h

Define la estructura encargada de representar cada espacio individual del mapa del juego.

Cada celda funciona como el contenedor de los 3 elementos esenciales del sistema:

- Terreno: determina características ambientales
- Unidades
- Edificios: indicando su ocupación actual

Gracias a esta estructura, el juego puede controlar la base física sobre la cual se desarrolla toda la dinámica táctica de la partida.

Controlador.h

Define la jerarquía de controladores encargados de manejar la fase activa de cada facción durante el turno.

Sus funciones principales son:

- resolver_fase: ejecuta las acciones automáticas de la facción.
- aplicar_rutina: determina cómo responde a los eventos del mundo.

A partir de esta base se derivan tres controladores específicos:

1. El del jugador
2. El del sistema enemigo (IA)
3. El neutral.

Este diseño permite que cada facción tenga su propio comportamiento.

Contexto.h

- Administra el estado global del juego y sus sistemas principales. Actúa como el “centro de mando”.
- Controla el mapa, las facciones, los turnos, los puntos de acción, los eventos del mundo, el clima y las misiones.
- Revisa la economía global mediante el seguimiento territorial, del puntaje, las mejoras y las misiones activas.
- También expone operaciones esenciales como guardar/cargar partida, mostrar información relevante al jugador, ejecutar eventos del mundo y detectar condiciones de derrota.

Evento.h

- Automatiza situaciones que NO dependen del jugador.
- Permite que el mundo tenga “vida propia”: refuerzos, clima, ataques, efectos globales.
- Cada evento incluye:
  - tipo → qué clase de suceso es (Clima, Refuerzo, etc.)
  - turno_activacion → cuándo ocurre
  - ejecutar() → acción concreta sobre el Contexto
- Método clave:
  - `bool debe_activarse(int turno_actual) const;`

    Permite al motor del juego activar eventos automáticamente.

1. EventoRefuerzo
   - Inserta unidades en el mapa.
   - Simula llegada de tropas externas.
2. EventoClima
   - Afecta moral.
   - Cambia condiciones globales.

Evento.h convierte al juego en un sistema dinámico con un “calendario” de sucesos automáticos.

Faccion.h

- Representa a cualquier participante del juego (jugador o sistema) gestionando su economía, moral y territorio.
- Administra los recursos (comida, metal, energía).
- Controla el parámetro psicológico moral, afectado por clima o eventos.
- Registra qué territorios controla la facción mediante una lista de coordenadas.
- Sobrecarga de operadores:
  - += añade territorio conquistado de forma elegante.
  - > compara facciones según dominio territorial → clave para condiciones de victoria.
- Evalúa capacidad de pago con:
  - `bool puede_pagar(const Recursos& costo) const;`

Coordenada.h

Define la estructura fundamental para ubicar cualquier cosa en el tablero 2D del juego

- Estandariza cómo se representan posiciones dentro del mapa (fila, columna).
- Permite que diferentes módulos (IA, movimientos, edificios, combate, eventos) hablen un lenguaje común de coordenadas.
- Constructor flexible: permite comenzar en (0,0) o cualquier ubicación.
- Sobrecarga de operadores == y != para comparar posiciones sin necesidad de funciones externas.
- Sobrecarga del operador << → facilita depurar e imprimir posiciones en consola sin código repetido.
- Toda acción que dependa de “dónde” ocurre —movimiento, captura, dominio, eventos, clima— se basa en esta clase.

Edificio.h

Define la estructura general de todos los edificios del juego. Contiene la clase base Edificio, que establece los atributos esenciales (tipo y propietario) y declara dos funciones clave que todos los edificios deben implementar: efecto_turno, que aplica la producción o beneficio que el edificio otorga cada turno, y reaccion_evento, que define cómo responde el edificio ante eventos especiales del mundo. Además, este archivo declara las clases derivadas Granja, Cuartel, Torre y Forja, cada una representando un edificio distinto con comportamientos específicos, manteniendo así la arquitectura polimórfica del sistema.

Explicación de archivos fuente(.cpp)

Unidad.cpp

Implementa el comportamiento real de todas las unidades del juego. Aquí se definen las acciones que cada unidad puede realizar durante la partida. El archivo contiene la lógica que permite mover, atacar y activar habilidades especiales, utilizando polimorfismo para que cada tipo de unidad (soldado, arquero, caballero, mago e ingeniero) ejecute su propia versión de estas acciones.

Mapa.cpp

Se encarga de construir y mostrar el escenario donde ocurre la partida. Aquí se genera la matriz de celdas que forman el mapa y se asignan los distintos tipos de terreno (llanura, bosque, pantano, agua, montaña) según las coordenadas.

Controlador.cpp

Define cómo actúan las facciones durante su turno y cómo responden a los eventos del mundo.

Incluye tres controladores:

- Jugador: gestiona la fase del jugador y sus reacciones a eventos.
- Sistema (IA): realiza ofensivas, patrullajes, defensa y genera guardianes automáticamente.
- Neutral: maneja efectos sobre estructuras sin dueño.

Este módulo convierte la lógica del contexto en acciones reales y permite simular un mundo dinámico y coherente dentro del ciclo de turnos.

Contexto.cpp

Implementa el núcleo del juego:

- Administra el estado global, coordina el mapa, las facciones, los recursos, el clima, las misiones, los turnos y los puntos de acción.
- Controla los eventos automáticos (refuerzos, clima, sabotajes), la captura de edificios neutrales, la verificación de victoria/derrota y el progreso de la partida.

Además, registra acciones por turno, gestiona guardado/carga y genera reportes finales.

Es el módulo que asegura coherencia y que todo el juego funcione como un sistema integrado.

Evento.cpp

Evento.cpp es el archivo donde se ejecutan los eventos automáticos del juego, o sea, las cosas que pasan solas en ciertos turnos. Aquí se manejan dos tipos de eventos: los refuerzos y el clima. Cuando toca un refuerzo, el juego recorre el mapa y va colocando unidades nuevas en las casillas libres según la facción que las recibe; esto sirve para que el sistema gane tropas sin que el jugador haga nada. Y cuando ocurre un evento de clima, se aplican efectos generales en la moral: una tormenta baja la moral de todos y un clima despejado sube la moral del jugador. Estos cambios afectan cómo pelean las unidades. En resumen, Evento.cpp controla todo lo inesperado del juego: cuándo aparecen tropas nuevas y cómo cambia el clima, dejando todo registrado en la bitácora del turno.

Edificio.cpp

Implementa los comportamientos definidos en el encabezado. Cada edificio aporta recursos diferentes a la facción que lo controla a través del método efecto_turno:

- La Granja produce comida,
- El Cuartel genera metal,
- La Torre otorga energía,
- Y la Forja incrementa la producción de metal de manera más fuerte.
- Las funciones reaccion_evento están definidas por ahora como vacías, dejando preparado el sistema para reaccionar ante eventos más avanzados en el futuro. En conjunto, este archivo convierte a los edificios en elementos activos del sistema económico, encargados de sostener los recursos del jugador durante la partida.

Probamos el código

Gracias {

}