

Universidad de Ingeniería y Tecnología



Documento Técnico: Arquitectura del código del proyecto “Rival Frontiers”

ASIGNATURA

Programación II

LENGUAJE

C++

DOCENTE

Rubén Rivas

INTEGRANTES

Tamara Lanfranco Perez

Jhon Adrian Urrunaga Rojas

Ricardo André Chávez Rosas

Lima-2025

1. Arquitectura de Datos y Estructuras Fundamentales

El diseño del proyecto se basa en la programación orientada a objetos (POO) para crear el mundo del juego, utilizando herencia y polimorfismo, lo cual permite manejar diversos tipos de entidades y componentes.

Contexto Global:

La clase Context , que se encuentra dentro del archivo proyecto.h, actúa como el contenedor global del estado del juego, centralizando la gestión de:

- **Recursos:** Instancias de la clase Recursos (comida, metal, energía) para el Jugador y el Sistema.
- **Controles:** Un vector de punteros a la clase base controles para orquestar el ciclo de turnos.
- **Entidades:** Vectores que contienen punteros a Unidad y Evento.
- **Turno:** Un contador entero para el número de turno actual.

```
class controles {  
public:  
    virtual void update(class Context&) =0;  
    virtual string name() const = 0;  
    virtual ~controles(){}  
};
```

```
class Entidad {  
private:  
    Coord posicion;  
    Owner owner;  
public:  
    Entidad(Coord c, Owner o);  
    virtual ~Entidad() {}  
    Coord getPosicion() const;  
    Owner getOwner() const { return owner; }  
    void setPosicion(const Coord& c) { posicion = c; }  
};
```

```
class Context{  
public:  
    vector<Unidad*> units;  
    vector<Evento*> events;  
    vector<controles*> controllers;  
  
    Recursos playerResources;  
    Recursos systemResources;  
    int turn;  
  
public:  
    Context();  
    ~Context();  
    void addUnit(Unidad* u);  
    void addEvent(Evento* e);  
    void addController(controles* c);  
};
```

El Mundo del Juego (Mapa y Celda)

El tablero se modela mediante la clase Mapa que se encuentra en Mapa.h , que contiene una matriz bidimensional de objetos Celda cuya clase se encuentra en Clases_baseMapa.h.

- Celda encapsula el contenido de una posición específica: un puntero a Terreno, un puntero a Edificios, un puntero a Unidad, y el Owner (Propietario: PLAYER, SYSTEM, NEUTRAL).
- La lógica de dominio territorial se resuelve en Mapa mediante el método int calcularDominio(Owner p) const.

```

using namespace std;

class Mapa {
private:
    int filas;
    int columnas;
    vector<vector<Celda>> matriz;

public:
    Mapa(int _filas = 12, int _columnas = 12);

    int get_Filas() const { return filas; }
    int get_Columnas() const { return columnas; }

    Celda& get_Celda(int x, int y);
    void setTerreno(int x, int y, Terreno* t);
    void setEdificio(int x, int y, Edificios* e);

    bool esCoordenadaValida(int x, int y) const;
    void imprimir() const;

    void operator+= (const pair<Coord, Owner>& p);
    int calcularDominio(Owner p) const;
    friend ostream& operator<<(ostream& os, const Mapa& m);
};

```

2. Módulos de POO (Herencia y Polimorfismo)

El código implementa jerarquías de clases abstractas que permiten un manejo polimórfico de las entidades clave:

Módulo	Clase Base Abstracta	Clases Derivadas	Uso Polimórfico
Unidades	Unidad En unidades.h	Soldado, Arquero, Caballero, Mago	atacar(), habilidad_especial()
Terrenos	Terreno En Terreno.h	Llanura, Bosque, Agua, Montaña	costo_movimiento(), bono_defensa()
Edificios	Edificios En Edificios.h	Granja (Gr), Torre (To), Cuartel (Cu)	efecto_turno(), reaccion_evento()

Controladores	controles En proyecto.h	JugadorControl, SystemController, NeutralController	update() (Ciclo de Turno)
Eventos	Evento En proyecto.h	RecursosEvento	apply() (Efecto del Evento)

Esta estructura garantiza que se puedan añadir nuevos tipos de unidades, terrenos o edificios sin tener que realizar grandes modificaciones al código base, esto ayuda a cumplir el requisito de Herencia y Polimorfismo.

3. Flujo de Juego y Utilidades

Ciclo de Turno (Controladores)

El ciclo de juego por turnos se realiza a través de la interfaz polimórfica de controles.

1. El objeto Context mantiene una lista de controles (Jugador, Sistema, Neutral).
2. En cada ronda, el método update(Context&) es llamado secuencialmente para cada controlador, ejecutando la lógica que corresponde para cada facción o sistema.

Sobrecarga de Operadores

Para mejorar la utilidad y la experiencia del usuario (reportes y visualización), se empleó la sobrecarga de operadores:

Clase	Operador Sobrecargado	Función
Mapa	operator<< (Salida)	Impresión detallada y codificada del mapa [XY].
Recursos	operator<< (Salida)	Reporte conciso del estado de los recursos.

Mapa	operator+=	Asignación directa de propiedad a una celda.
Recursos	operator> (Comparación)	Lógica para determinar si un set de recursos es "mayor" que otro.

Visualización y Reportes (Experiencia del Usuario)

La impresión del mapa mediante operator<< se realizó para ofrecer un reporte compacto en la consola:

Componente	Código Abreviado
Propietario (X)	P (Player), S (System), N (Neutral)
Contenido (Y)	Primer carácter de la unidad (S de Soldado), Edificio (G de Granja), o Terreno (L de Llanura).

El sistema de registro Logger proporciona una herramienta de persistencia opcional y de *logging* para auditoría de eventos del juego.

Conclusión

Nuestro código demuestra un diseño modular y que permite agregar nuevos códigos para extenderlo. Aplicamos patrones de POO (polimorfismo en 4 dominios principales), usamos estratégicamente la sobrecarga de operadores para la interfaz y centralizamos el estado del juego en Context para establecer una base capaz de facilitar la expansión de la lógica del juego.