Fase 2 — Simulador de la Batalla por Centauro

Fundamentos de Programación (213022)
UNAD — ECBTI

Universidad	Universidad Nacional Abierta y a Distancia — UNAD
Escuela	ECBTI — Escuela de Ciencias Básicas, Tecnología e Ingeniería
Curso	Fundamentos de Programación (código 213022)
Fase	Fase 2 — Variables, constantes y estructuras de control
Estudiante	Juan David Salas Camargo
Identificación	1049650019
Grupo	213022_30
Programa académico	Ingeniería de Sistemas
Fecha de entrega	03-11-2025

Situación problema seleccionada

Estudiante	Juan David Salas Camargo
Problema asignado	Problema 3 — Simulación de la batalla en el planeta Centauro

Descripción breve del problema:

El planeta Centauro se encuentra en guerra. Los ejércitos del bien y del mal están conformados por razas con valores comprendidos entre 1 y 5. Se requiere un programa que permita configurar la cantidad de integrantes por raza, calcule la fuerza total de cada ejército y determine si el bien gana, el mal gana o se produce un empate. Los ejemplos de la guía (1 Osito pierde contra 1 Hoggin, 2 Ositos empatan contra 1 Hoggin, 3 Ositos ganan a 1 Hoggin) sirven como casos de referencia para validar la solución.

3.1 Razas y valores utilizados

Bando benévolo	Valor	Bando malvado	Valor
Ositos	1	Lolos	2
Príncipes	2	Fulanos	2
Enanos	3	Hoggins	2
Caris	4	Lurcos	3
Fulos	5	Trollis	5

Tabla 2. Requerimientos funcionales

ID	Descripción	Entradas	Resultados (salidas)
R1	Registrar la cantidad de unidades para cada raza benévola.	Nombre de la raza benévola, número de unidades.	Roster almacenado para el ejército del bien.
R2	Registrar la cantidad de unidades para cada raza malvada.	Nombre de la raza malvada, número de unidades.	Roster almacenado para el ejército del mal.
R3	Calcular el poder total de cada ejército sumando valor de raza por cantidad.	Roster del bien y del mal.	Totales de unidades y poder acumulado.
R4	Determinar el resultado de la batalla comparando los poderes.	Poder total del bien y del mal.	Resultado textual: gana el bien, gana el mal o empate.
R5	Presentar el resultado y permitir gestionar la simulación desde consola o GUI.	Acciones del usuario (menú, botones).	Reportes en consola, ventanas GUI y persistencia en data/armies.json.
R6	Ofrecer créditos del proyecto y ambientación musical opcional.	Solicitud del usuario (menú o botón).	Ventana emergente o listado en consola con créditos; música 8 bits generada procedimentalmente.

Análisis y diseño

Paradigma empleado: programación estructurada con separación por capas (controladores, servicios, infraestructura y UI).

Modelo de datos: razas parametrizadas con valor base; ejércitos almacenan pares (raza, cantidad). El archivo data/armies.json conserva la configuración entre sesiones.

Persistencia: clase JsonArmyStorage en app/infrastructure/persistence.py.

Lógica de batalla: servicio BattleService calcula los poderes y define el resultado (BattleOutcome).

Interfaz de usuario: ConsoleUI y GameWindow comparten el mismo GameController. Se añadió un botón/menú para créditos y música procedimental en la GUI.

Música procedimental: ProceduralChiptune genera temas inspirados en RPG clásicos utilizando progresiones armónicas y escalas pentatónicas.

Diagrama de flujo: a continuación.

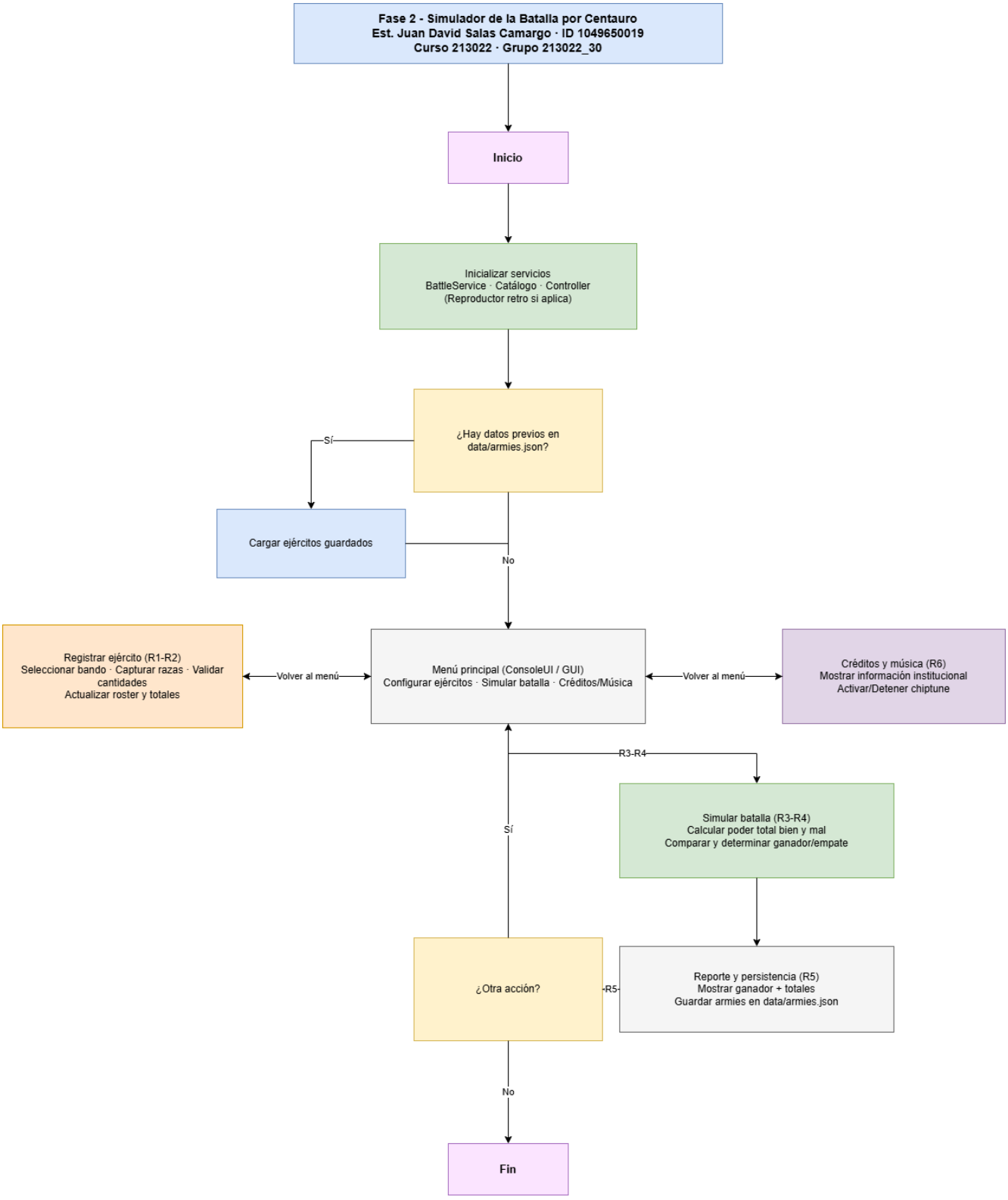

Figura 1. Diagrama de flujo del simulador de batalla (versión de entrega).

Implementación

Archivo principal: main.py — soporta modos consola, GUI o menú interactivo, y la bandera --no-music.

Controlador: app/controllers/game_controller.py — coordina servicios, persistencia, créditos y música.

Servicio de batalla: app/services/battle_service.py — aplica la lógica de comparación de fuerzas.

Interfaces: app/ui/console.py (menú textual) y app/ui/gui.py (Tkinter); ambos acceden al mismo controlador.

Música: app/infrastructure/music.py — generador 8 bits y reproductor pygame.

Pruebas automatizadas: python -m unittest (ver carpeta tests/).

Pasos de ejecución: 1) Crear (opcional) entorno virtual; 2) Instalar dependencias opcionales; 3) Ejecutar modo consola/GUI; 4) Ver créditos.

Pruebas realizadas

Caso evaluado	Configuración (Bien vs Mal)	Resultado esperado	Resultado obtenido
Caso guía 1	1 Osito vs 1 Hoggin	Gana el mal	Gana el mal
Caso guía 2	2 Ositos vs 1 Hoggin	Empate	Empate
Caso guía 3	3 Ositos vs 1 Hoggin	Gana el bien	Gana el bien
Prueba automatizada	python -m unittest	Todas las pruebas pasan	OK

Conclusiones

1. La simulación demuestra que las variables, estructuras de control y colecciones permiten modelar escenarios complejos a partir de reglas sencillas.

2. La separación en capas facilita reutilizar la lógica en varias interfaces (consola y GUI) y permite añadir valor agregado (música y créditos) sin modificar el núcleo del problema.

Referencias bibliográficas

- Python Software Foundation. Python 3.12 Documentation. https://docs.python.org/3/

- Pygame Community. Pygame Documentation. https://www.pygame.org/docs/

- Universidad Nacional Abierta y a Distancia (UNAD). Guía de aprendizaje — Fase 2. Variables, constantes y estructuras de control. 2025.