

Computación I , Universidad de Mendoza, Facultad de Ingeniería

# Presentación de RPG

## DIVINE LIGHT

---



---

## **Introducción:**

Divine Light es un RPG programado en python y ejecutado a través de la terminal. La característica principal del juego es la toma de decisiones respecto a la historia, teniendo muchos caminos posibles para que cada partida se sienta diferente. Dentro de cada partida hay encuentros con diferentes enemigos, un sistema de niveles y diferentes personajes jugables, cada uno con sus características.

Divine light utiliza una estructura de árboles para cargar la historia y la toma de decisiones, las mismas se cargan a través de un JSON donde cada hijo es una decisión posible. Dentro de cada nodo existe la posibilidad de encontrar un enemigo y que se muestre un mapa en la consola. El menú y la interacción con el mismo está realizado con la biblioteca rich terminal y pyinput respectivamente. El resto se utilizan ciertos métodos para mejorar la inmersión y simular mecánicas propias de los RPG.

## **Desarrollo**

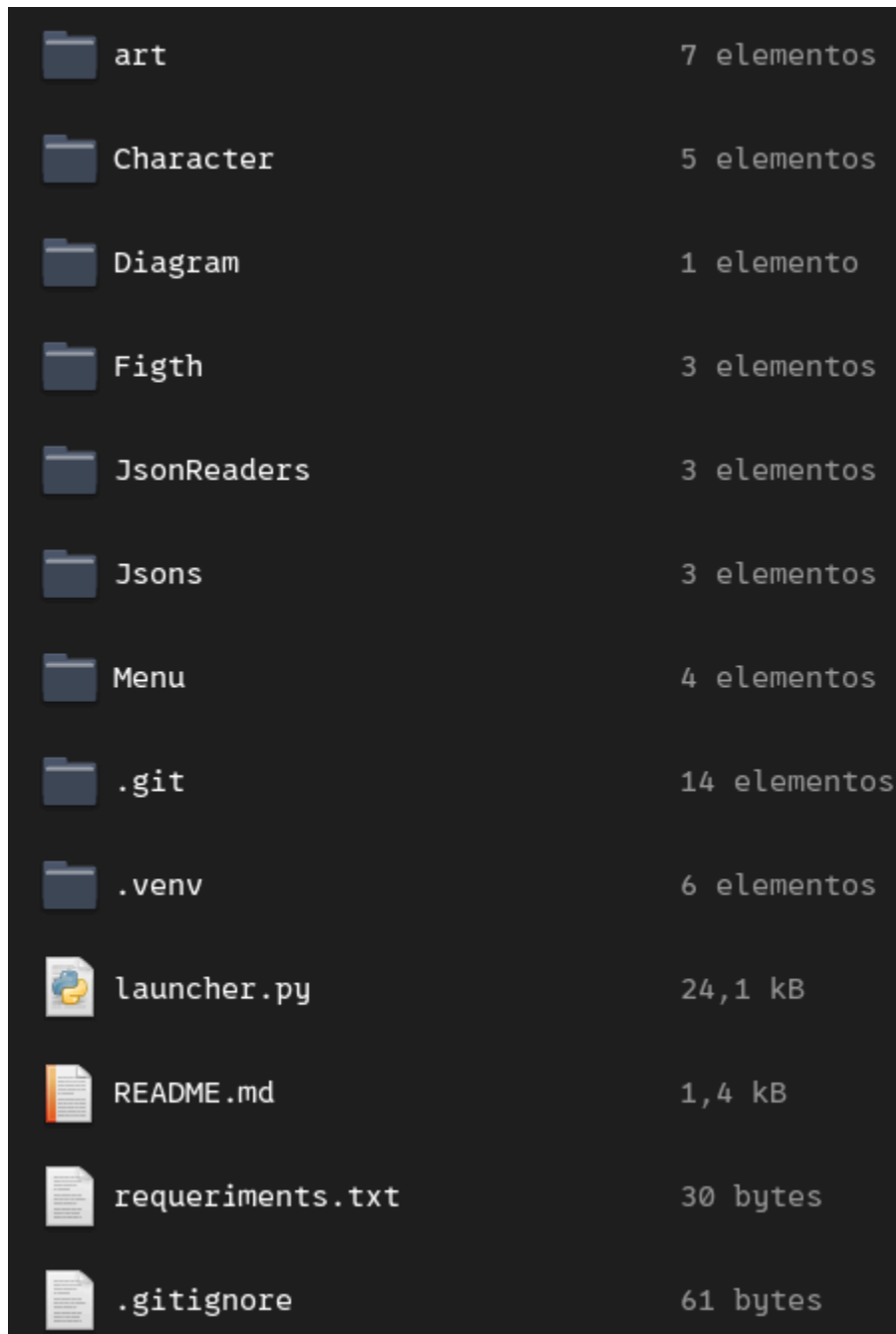
Divine light comenzó como un proyecto académico, que lentamente fue tornándose en algo más ambicioso y de trabajo en equipo. Se busco la realización de un videojuego RPG funcional y bien ambientado, con buena narrativa, sistema de combate divertido pero desafiante y fácilmente escalable.














Con esos argumentos por delante, nace “Divine Light” , inspirado en obras como Dark Souls, Berserk o Fear and Hunger. La idea central es generar una sensación de progresión constante, donde el jugador atraviesa zonas decadentes y desoladas , enfrentando enemigos y tomando decisiones que modifican su recorrido narrativo, esto se logró gracias a:

- 1) Inmersión a través del texto y la ambientación: Aunque no hay gráficos animados, se detallan las descripciones para volverlas ricas en tono y atmósfera. El texto no sólo narra sino que transmite emociones y tensión.
- 2) Sistema narrativo ramificado: Cada fragmento de historia es independiente dentro de un archivo JSON, de esta manera las decisiones tomadas por el jugador puede llevarlo a distintos escenarios u otras zonas, sin necesidad de tocar código.
- 3) Progresión del personaje: El jugador gana experiencia, sube de nivel, mejora atributos y enfrenta situaciones y combates cada vez más complejas, dando una recompensa al jugador por seguir en la historia

---

Estructura del proyecto:



 art	7 elementos
 Character	5 elementos
 Diagram	1 elemento
 Figth	3 elementos
 JsonReaders	3 elementos
 Jsons	3 elementos
 Menu	4 elementos
 .git	14 elementos
 .venv	6 elementos
 launcher.py	24,1 kB
 README.md	1,4 kB
 requeriments.txt	30 bytes
 .gitignore	61 bytes

Lo que se intentó buscar con esta arquitectura y lógica, es la fácil distribución de funcionalidades además de ser fácilmente reconocibles, siendo [launcher.py](#) el orquestados del resto de componentes.

---

## Lógica

Una de las decisiones más importantes al momento de la implementación fue que los eventos, enemigos y decisiones no están programados dentro del código, sino que se leen dinámicamente desde archivos JSON. Cada archivo representa una zona del juego, y dentro de él se definen múltiples nodos. Cada nodo tiene la posibilidad de ser:

- Historia: Contiene texto descriptivo y una lista de opciones, estas últimas son las que el jugador decide para moverse a otro nodo

```
{  
  
  "id": "puente_roto",  
  
  "tipo": "historia",  
  
  "descripcion": "Cruzás el puente cubierto de ceniza, sintiendo el eco del metal corroído bajo tus pies.",  
  
  "opciones": [  
  
    { "texto": "Avanzar hacia la catacumba", "resultado": "#entrada_catacumba" },  
  
    { "texto": "Regresar", "resultado": "FIN" }  
  
  ]  
  
}
```

- Combate: Se define al enemigo, la descripción del enfrentamiento y según el resultado de la batalla a donde se direcciona el jugador.

```
{  
  
  "id": "encuentro_guardian",  
  
  "tipo": "combate",  
  
  "descripcion": "Un Guardián emerge entre las cenizas, blandiendo una espada incandescente.",  
  
  "enemigos": [  
  
  ]  
  
}
```

---

```
{  
  "nombre": "Guerrero de Ceniza",  
  "ascii": "guardian.txt",  
  "stats": { "LEVEL": 1, "HP": 60, "ATK": 14, "DEF": 8, "AG": 6 },  
  "xp": 25  
}  
],  
"victoria": "#despues_del_guardian",  
"derrota": "FIN"  
}
```

De esta forma nuestro motor narrativo no necesita lógica condicional compleja, sino que basta con leer el tipo de nodo y el resultado para seguir ejecutándose.

Todo esto se hace dentro de `JsonsReader`, que es la columna vertebral del motor narrativo, donde sus funciones principales son `load_zone()` (carga el archivo y genera los índices), `get_current_node_()` (devuelve el nodo actual) y `jump_to_by_index()/jump_to_result()` (cambian de nodo según lo seleccionado o sucedido en combate).

## Personajes

`Character` actúa como clase base, definiendo los atributos comunes como vida, ataque, defensa, etc.

`Player` hereda de `Character` y agrega comportamiento específico del jugador como subir de nivel

`Enemy`, también derivado de `Character`, incluye el método `from_json()` que permite instanciar enemigos directamente desde los datos del nodo

## Sistema de combate

---

El combate se activa automáticamente al entrar a un nodo de tipo combate. La lógica se encuentra dentro de `run_game()` en el archivo `launcher.py`, y funciona así:

Se crea una copia temporal del jugador (`player_battle`) para evitar alterar el objeto principal mientras se desarrolla el combate, luego, se instancia el enemigo mediante `Enemy.from_json()`.

Se ejecuta la función `Prioridad_Ataque()`, que determina quién ataca primero (comparando velocidad y precisión), se calculan los daños, críticos y respuestas, después La función `Control Vida()` revisa si alguno de los participantes murió, si el enemigo muere: el jugador gana XP, puede subir de nivel si llegaste a mas de 100 de XP y se muestra a continuación un menú en donde el usuario puede subir los atributos a su antojo, y se ejecuta el salto narrativo a victoria, si el jugador muere: se salta a derrota o FIN.

El combate tiene además un registro visual (`combat_log`) que muestra los últimos eventos del turno, todo dentro de paneles Rich.

## Sistema de Niveles

Cuando el jugador acumula 100 puntos de experiencia, se activa el evento de level-up.

Se ejecutan los siguientes pasos:

- Se incrementa el nivel del jugador en +1.
- Se agregan automáticamente +10 puntos de vida máxima.
- Se entregan 3 puntos de atributos a distribuir.

Se abre un menú Rich interactivo para asignar puntos en ataque, defensa, magia o precisión.

Al finalizar la asignación, el juego continúa normalmente. Esto permite que el jugador

Este diseño mantiene la progresión clara y natural, sin romper la inmersión ni requerir inputs de texto.

## Interfaz

Toda la interfaz se maneja mediante dos librerías:

---

Rich: usada para renderizar paneles, tablas y textos con color, borde y formato. Permite construir una interfaz legible, atractiva y dinámica dentro de la consola.

pynput: permite capturar las teclas en tiempo real sin bloquear la ejecución. Se usa en todos los menús y en el game loop principal, manejando ↑, ↓, Enter y Esc.

Un detalle importante del diseño fue separar los listeners de teclado por contexto (menú principal, selección de personaje, juego), esto evita errores comunes donde múltiples listeners activos capturan la misma entrada, generando comportamientos erráticos.

## Escalabilidad

Lo más importante en este aspecto del juego, es el sistema para crear contenido nuevo, ya que solamente hace falta seguir la estructura de los JSON , tanto para la historia como para los enemigos, haciendo que sea bastante fácil poder estructura cualquier tipo de historia de un juego formato RPG.

## Trabajo futuro y conclusiones

El código actual establece una base sólida, pero existen múltiples caminos de evolución posibles:

- Sistema de guardado y carga de partida.
- Implementación de inventario, ítems y equipamiento.
- Habilidades especiales por clase de personaje (magia, ataques cargados, efectos de estado).
- Balance dinámico de enemigos según nivel del jugador.
- Interfaz visual más compleja con Rich Layout o incluso una migración a una interfaz gráfica (Pygame, Tkinter, etc.)
- Sistema de diálogos con variables para permitir decisiones más profundas y finales alternativos.

El desarrollo de “Divine Light” representó un trabajo de integración entre programación, diseño narrativo y estructura de datos.

---

El proyecto logró consolidar un motor narrativo funcional, un sistema de combate por turnos estable y una experiencia inmersiva dentro de una interfaz de texto, todo sostenido sobre una arquitectura limpia y extensible.

Cada componente del sistema cumple un rol claro: los JSON definen el mundo, el lector lo interpreta, los menús y Rich lo muestran, y el jugador lo recorre.

La decisión de adoptar una estructura modular y un flujo basado en loops independientes con sus propios listeners fue clave para garantizar la estabilidad del juego y la posibilidad de ampliarlo sin problemas.

En definitiva, el proyecto combina una ejecución técnica sólida con una visión narrativa coherente, y deja la puerta abierta para seguir expandiendo este universo digital con nuevos contenidos, mecánicas y desafíos.

Es, en esencia, un punto de partida firme para un RPG completo, construido desde cero con Python y con un enfoque de ingeniería que prioriza la claridad, la escalabilidad y la jugabilidad.