DOCUMENTACIÓN II PROYECTO.

Estudiantes.
1. Jacky Fang Zhang - 2023121133
2. Emmanuel García Quiros - 2025065962

Grupo: 2
Curso: Introducción a la Programación
Semestre: II del 2025

<u>Atributos de Análisis de Problema:</u>

Problema complejo de ingeniería.

En este proyecto se aborda el problema complejo de diseñar y simular un sistema de escape y persecución dentro de un laberinto generado de forma aleatoria, donde un jugador con una cantidad limitada de energía debe tomar decisiones de movimiento eficientes mientras interactúa con cazadores controlados por algoritmos de búsqueda y evasión. El problema requiere principios de matemáticas discreta para representar el mapa como una matriz y garantizar la existencia de caminos válidos, así como conceptos básicos de programación para modelar el comportamiento del jugador y los enemigos. Además, se integran aspectos de desarrollo sostenible al considerar el uso eficiente de recursos, la reutilización y mantenibilidad de código, y el diseño de una herramienta que sirva como apoyo educativo en la comprensión de algoritmos y toma de decisiones.

Análisis contexto y variables relacionados con el problema complejo identificado.

El contexto de nuestro problema es el diseño de un videojuego sencillo donde un jugador intenta escapar o cazar dentro de un laberinto que cambia en cada partida, y todo eso tiene que funcionar y ser entendible para alguien que apenas empiece a programar. En este escenario intervienen varias variables importantes: la matriz que representa el mapa, la posición del jugador y de los enemigos, el tipo de terreno en cada celda (camino, muro, túnel o liana), la cantidad de energía posible y disponible, la dificultad elegida y el tiempo de juego, porque todo eso afecta si realmente existe un camino posible y qué tan justo se sienta el juego. Desde el punto de vista de sostenibilidad, tratamos de que el sistema sea sostenible de dos principales formas, primero, en el uso de recursos que nos de python para poder elaborar una representación sencilla del laberinto, para que el juego pueda correr en equipos modestos sin desperdiciar mucha memoria virtual (RAM) ni procesador. En segundo lugar, en la sostenibilidad del propio código, separando responsabilidades en varios módulos (archivos .py), para que sea fácil de mantener, reutilizar, revisar y extender en futuros cursos o proyectos.

Plan Solución.

Se propone como plan de solución implementar el problema mediante un videojuego dividido en dos modos (Escapar y Cazador), usando una matriz para representar el laberinto y un conjunto de clases para jugador, enemigos, terrenos, puntajes y configuración del juego. En cada partida se genera un mapa aleatorio verificando que haya al menos un camino válido, y sobre ese mapa se aplican reglas claras de movimiento, consumo de energía, uso de trampas y comportamiento de los enemigos, de manera que el sistema simule las decisiones de escape o persecución bajo recursos limitados.

Desde el enfoque de sostenibilidad, el plan busca que la solución sea ligera y mantenible: se usan estructuras simples (matrices y archivos de texto) y algoritmos básicos para que el juego corra en equipos con pocos recursos sin problema, y se organiza el código en módulos reutilizables que facilitan extender o mejorar el proyecto en futuros cursos sin tener que comenzar de 0.

Pros y contras de las soluciones planteadas.

En general, la solución que se plantea tiene varios puntos a favor: al usar una matriz para representar el laberinto y clases separadas para jugador, enemigos y terrenos, el sistema es claro, modular, y relativamente fácil de mantener o mejorar en un futuro, lo cual aporta a la sostenibilidad y longevidad del código. Además, la generación aleatoria de mapas con verificación de caminos, el sistema de energía, las trampas y los modos de juego permiten explorar distintas estrategias de escape y persecución usando recursos limitados, algo valioso tanto desde la ingeniería como desde lo educativo.

Como contras, la solución también tiene limitaciones, al usar algoritmos relativamente simples, la generación de laberintos, el comportamiento de los enemigos y el balance de dificultad pueden resultar un poco repetitivos. Mejorar eso implicaría mayor complejidad y mayor conocimiento en programación y de diseño. Por otro lado, aunque el uso de estructuras sencillas favorece el rendimiento y la sostenibilidad técnica, también restringe el tipo de análisis que se puede hacer.

Atributo de herramientas de Ingeniería:

Para resolver el problema complejo de desarrollo de un videojuego sobre escape y persecución en un laberinto, con inteligencia artificial básica, generación procedural de mapas y gestión de estados en tiempo real, utilizamos:
Python como lenguaje base en la elaboración del proyecto

Pygame para gráficos 2D, gestión de sonido, eventos en tiempo real y loop principal del juego

Tkinter para interfaces de usuario(menús, configuración, resultados)

Random para generación procedural de mapas y posiciones aleatorias

OS para gestión de archivos y persistencia de datos

Time para control de temporizadores y medición de performance

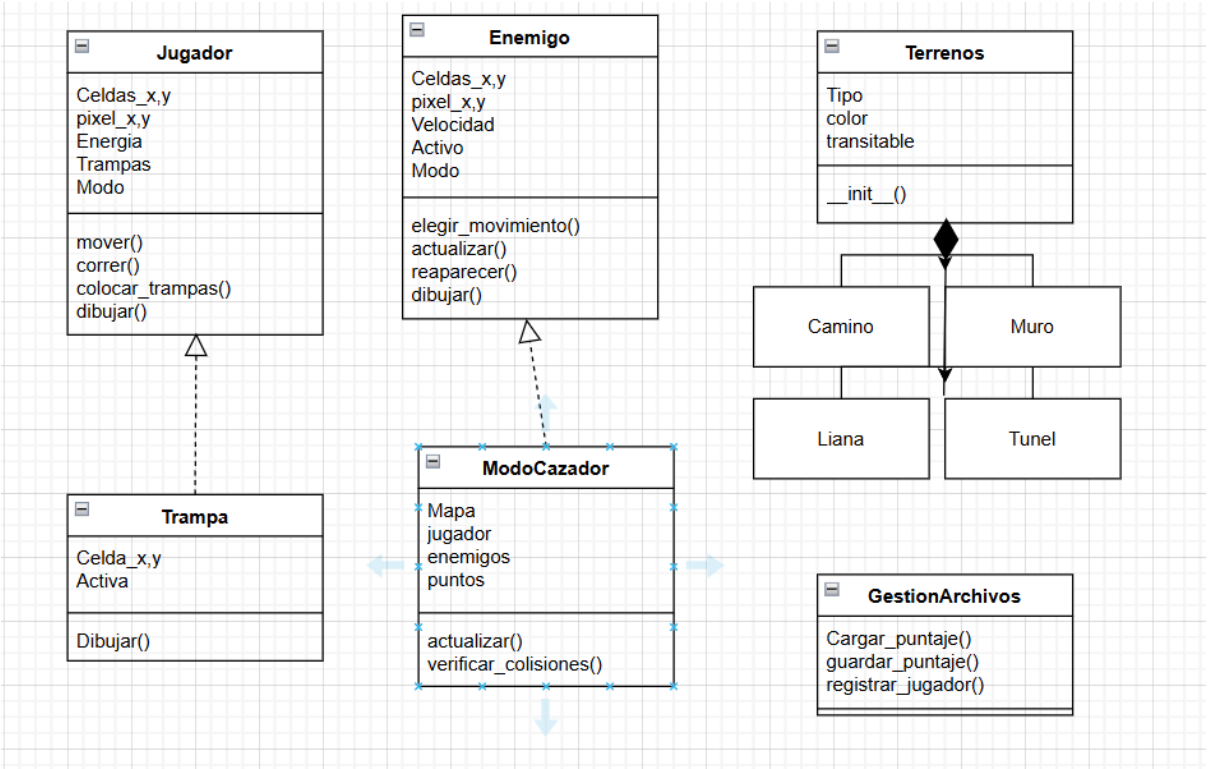
En la arquitectura y paradigmas se utilizó:

Programación Orientada a Objetos (POO) para modelar entidades del juego

Una arquitectura modular, en la que cada característica está separada en archivos diferentes para una mayor organización.

y se gestionó el proyecto mediante github, para un control colaborativo y distribución igual de avances, al igual de uso constante de commits con mensajes distributivos.

Diagrama de clases



Aplicación de técnicas y métodos

Generación procedural de mapas:

```
# Algoritmo implementado en generar_mapa_valido()

def generar_mapa_valido():

    while True:

        mapa = generar_mapa()  # Distribución aleatoria de terrenos

        inicio, salida = elegir_puntos_inicio_y_salida(mapa)

        if distancia(inicio, salida) < 10: continue  # Evitar mapas pequeños

        if es_resoluble(mapa, inicio[0], inicio[1], salida, visitado):

            return mapa, inicio, salida  # Garantizar camino válido
```

Sistema simple de IA para enemigos:

Modo Escapar: `elegir_movimiento_aleatorio()` con tendencia a perseguir al jugador

Modo Cazador: `elegir_movimiento_hacia_salida()` con sistema de puntuación por movimientos para mejor toma de decisiones

Pathfinding: Movimiento por celdas adyacentes con validación de transitabilidad

Guardado de puntos

```
# Sistema de archivos para puntajes y historial

def guardar_puntaje_escapa(nombre, puntos):

    puntajes = cargar_puntajes_existentes()

    puntajes.append((nombre, puntos))

    puntajes.sort(key=lambda x: x[1], reverse=True)

    guardar_top_5(puntajes)
```

Sistema de física y colisiones:

Movimiento por interpolación entre celdas

Detección de colisiones por coincidencia de coordenadas de celda

Sistema de energía con consumo y recuperación progresiva

Adaptaciones realizadas basadas en pruebas:

Movimiento del jugador:

Problema inicial: Movimiento completo entre celdas era muy lento

Solución: Implementamos movimiento parcial (media celda) para mayor agilidad

Código adaptado: iniciar_movimiento_medio() en clase Jugador

IA de enemigos en modo cazador:

Problema inicial: Comportamiento de huida era predecible

Solución: Sistema de puntuación por movimientos con múltiples prioridades

Código adaptado: elegir_movimiento_hacia_salida() con evaluación contextual

Generación de mapas:

Problema inicial: Mapas aleatorios podían ser irresolubles

Solución: Algoritmo de validación recursiva es_resoluble()

Código adaptado: generar_mapa_valido() con verificación de caminos

Gestión de dificultad:

Problema: Balance entre desafío y jugabilidad

Solución: Parámetros escalables por dificultad (velocidad, cantidad de enemigos, puntos)

Implementación: Variables condicionales basadas en configuracion.dificultad_actual