Instituto Tecnológico de Costa Rica

Campus San Carlos

Escuela de ingeniería en computación

Lenguajes de programación

Proyecto 2: Lenguaje Lógico y Orientado a objetos

Profesor

Oscar Mario Víquez Acuña

Estudiante

Joseph Fonseca Núñez

I Semestre, 2025

Documentación del Proyecto: Lenguaje Lógico y Orientado a objetos

1. Análisis del Problema

El problema fundamental para resolver, planteado en la Segunda Tarea Programada, consistía en el diseño y la implementación de un videojuego 2D de tipo "arcade" llamado "Tank-Attack". El reto principal no radicaba únicamente en el desarrollo de un juego funcional, sino en la integración de dos paradigmas de programación distintos: la Programación Orientada a Objetos (POO) y la Programación Lógica.

El objetivo era utilizar cada paradigma para resolver las partes del problema para las que son más adecuados. Los requisitos específicos del problema incluían:

- **Entorno del Juego (POO):** Se debía crear un motor de juego en tiempo real con una interfaz gráfica. Este motor debía gestionar múltiples objetos interactivos como tanques, balas y muros, así como los estados del juego (menús, niveles, fin de la partida).
- **Inteligencia Artificial (Programación Lógica):** La tarea más compleja de razonamiento, la búsqueda de rutas para los tanques enemigos debía ser resuelta exclusivamente con Prolog. Los tanques enemigos debían ser capaces de encontrar un camino hacia el tanque del jugador a través de un mapa con obstáculos.
- **Requisitos Funcionales:** El juego debía incluir:
    - Un máximo de 3 niveles con generación aleatoria de obstáculos.
    - Tres tipos de tanques enemigos con capacidades diferentes.
    - Dos tipos de objetivos a destruir, con diferente resistencia.
    - Un sistema de 3 vidas para el jugador por nivel.
    - Un módulo simple para la edición de pantallas.
    - Una experiencia de juego fluida y sin interrupciones.

El principal desafío técnico era, por tanto, crear un "puente" robusto y de alto rendimiento entre el motor de juego en Python (imperativo y orientado a objetos) y el motor de lógica en Prolog (declarativo).

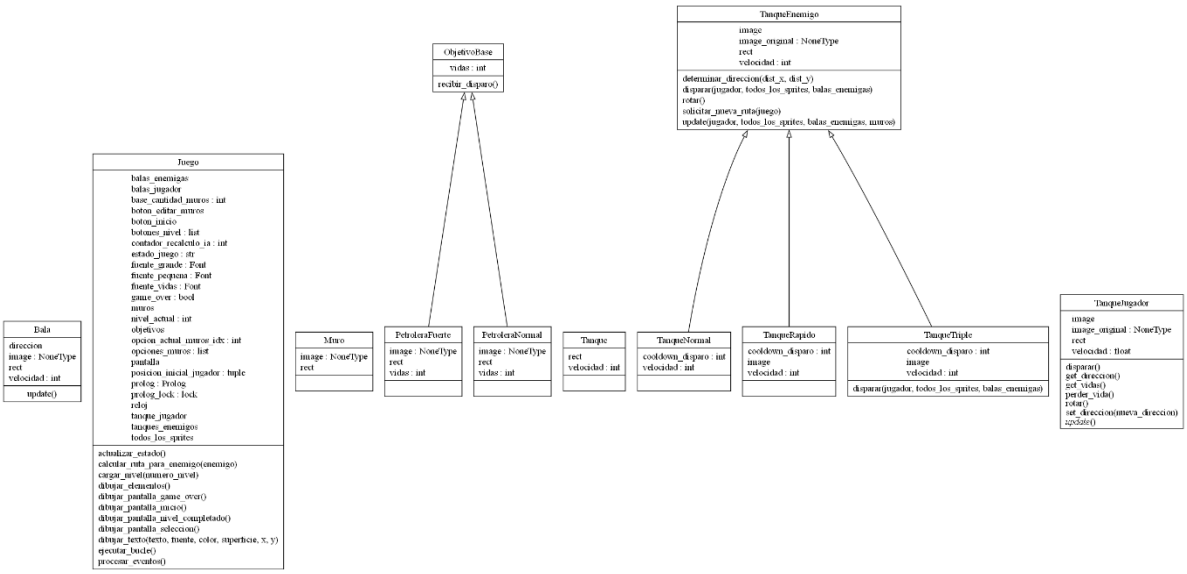
2. Análisis de la Solución Planteada

Para abordar el problema, se diseñó una arquitectura de software híbrida, separando claramente las responsabilidades según el paradigma más adecuado.

2.1. Arquitectura Orientada a Objetos (Python y Pygame)

Se eligió Python con la librería pygame para desarrollar el "frontend" y el motor principal del juego.

- **Estructura de Clases:** Se modeló cada entidad del juego como una clase, aplicando los principios de la POO. Se creó una clase base Tanque de la cual heredan TanqueJugador y TanqueEnemigo. A su vez, TanqueEnemigo actúa como clase padre para las variantes TanqueNormal, TanqueRapido y TanqueTriple, facilitando la reutilización de código y la creación de comportamientos especializados. Una estructura similar se aplicó para los objetivos.
- **Ocultamiento de Información:** Se implementó el ocultamiento de información mediante la convención de guion bajo (_) para atributos internos (ej. _vidas, _ruta), cumpliendo con el requisito del proyecto. El acceso a estos atributos se gestiona a través de métodos públicos (ej. get_vidas(), perder_vida()), lo que aumenta la robustez del código.
- **Gestión del Juego:** La clase Juego actúa como el orquestador principal, manejando el bucle del juego, los diferentes estados (menús, partida, etc.), la renderización de gráficos y la detección de colisiones.



2.2. Arquitectura Lógica (Prolog)

Para la IA de los enemigos, se utilizó un módulo en Prolog (pathfinding.pl) que se especializa en la búsqueda de rutas.

- **Algoritmo:** Se implementó un predicado buscar_camino/2 que ejecuta una **búsqueda en profundidad (DFS)**, tal como lo solicitaba el enunciado.
- **Optimización y Heurísticas:** Para hacer la búsqueda más eficiente y evitar caminos ilógicos, se implementaron dos mejoras clave:
    1. **Límite de Profundidad:** Se estableció una profundidad máxima de búsqueda para que el algoritmo descarte rutas excesivamente largas, evitando así cálculos infinitos.
    2. **Dirección Preferida:** Desde Python, se calcula la dirección general hacia el jugador y se pasa como una "pista" a Prolog. El algoritmo utiliza esta pista para explorar primero las direcciones más prometedoras, encontrando rutas más directas y eficientes sin dejar de ser una búsqueda en profundidad.

2.3. Integración y Solución de Rendimiento (Python + Prolog)

El principal obstáculo fue el bloqueo del juego durante las consultas a Prolog. Al ser una operación síncrona, detenía el bucle de Pygame, causando congelamientos.

- **Solución Final:** La solución definitiva, sugerida en el propio enunciado, fue implementar un sistema asíncrono mediante hilos (threading).
    - Cuando un enemigo necesita una ruta, no detiene el juego. En su lugar, lanza un hilo trabajador en segundo plano.
    - Este hilo se encarga de realizar la consulta a Prolog.
    - Mientras tanto, el hilo principal del juego sigue corriendo a 60 FPS, manteniendo una total fluidez visual y de control.
    - Cuando el hilo trabajador obtiene la ruta, se la asigna al tanque enemigo.
    - Se utiliza un threading.Lock para garantizar que solo un hilo acceda al intérprete de Prolog a la vez, evitando conflictos.

3. Análisis de Resultados Obtenidos

El producto final es un videojuego 2D completamente funcional que cumple con los requisitos planteados. El resultado de la solución implementada es una aplicación estable y fluida.

- Los tanques enemigos persiguen al jugador de manera efectiva, navegando por los laberintos generados aleatoriamente. Sus movimientos son suaves y sus ataques son coherentes.
- La integración de los dos paradigmas fue exitosa: el juego es interactivo y en tiempo real (gracias a POO y Pygame), mientras que las decisiones lógicas complejas son delegadas y resueltas eficientemente por Prolog.
- El problema de rendimiento fue completamente solucionado con la arquitectura multihilo, resultando en una experiencia de usuario sin interrupciones ni congelamientos.

- El juego presenta una jugabilidad variada gracias a los diferentes tipos de enemigos y objetivos, y a la generación aleatoria de los niveles.

4. Manual de Usuario

- **4.1. Requisitos de Instalación**
  - Python 3.x
  - Librería Pygame (pip install pygame)
  - SWI-Prolog
  - Librería Pyswip (pip install pyswip)
- **4.2. Instrucciones de Ejecución**
  - Navegar a la carpeta del proyecto a través de una terminal y ejecutar el comando: python main.py.
- **4.3. Controles del Juego**
  - **Movimiento:** Teclas W (arriba), A (izquierda), S (abajo), D (derecha).
  - **Disparo:** Barra Espaciadora.
  - **Navegación en Menús:** Clic del ratón.
- **4.4. Consejos para la mejor experiencia de juego**:
  - Ejecute el juego en el nivel 3 y con 40 muros, para que la búsqueda de datos de prolog no dure mucho.

5. Conclusiones

Este proyecto demuestra de manera práctica el poder y la flexibilidad de la programación multiparadigma. Se concluye que la Programación Orientada a Objetos es una herramienta excepcionalmente adecuada para modelar sistemas complejos con múltiples entidades que interactúan, como es el caso de un videojuego. Por otro lado, la Programación Lógica se revela como una solución elegante y potente para problemas de razonamiento simbólico y búsqueda, como el pathfinding.

El principal aprendizaje técnico del proyecto no fue el manejo de cada paradigma por separado, sino la resolución del desafío que supone su integración, específicamente en el ámbito del rendimiento. La implementación de una arquitectura asíncrona fue clave para lograr un producto final de calidad, demostrando que el diseño arquitectónico es tan importante como la lógica de la programación misma. El resultado es un sistema robusto donde cada paradigma es utilizado para resolver los problemas en los que sobresale.

6. Referencias

- **Python:** Lenguaje de programación principal. *https://www.python.org/*
- **Pygame:** Librería para el desarrollo de videojuegos en Python. *https://www.pygame.org/*
- **SWI-Prolog:** Intérprete de Prolog utilizado para la inteligencia artificial. *https://www.swi-prolog.org/*
- **Pyswip:** Librería de Python utilizada como puente para la comunicación con SWI-Prolog. *https://github.com/yuce/pyswip*