JUEGO DE LABERINTOS AUTÓNOMO EN PYTHON

Velasco Reyes Juan Ramón, Gallegos Alvarez Jovani, García Canteros Ángel Josué
{*jjf, gaaj010320, garciacanterosangeljosue057*}*@gs.utm.mx*
Profesor: M.C. Juan Juárez Fuentes

***Abstract*—Este artículo presenta un proyecto desarrollado en Python utilizando la biblioteca Pygame, que simula un juego de laberintos interactivo. Los laberintos se generan de forma procedural y los agentes, controlados por algoritmos, exploran el entorno dinámicamente para alcanzar un objetivo. El sistema incorpora modos de movimiento seleccionables, como movimientos aleatorios, sistemáticos y de búsqueda óptima. Se detalla la estructura modular del proyecto, organizada en cuatro componentes principales: `maze_generator`, `maze_app`, `entities` y `cons`. Este enfoque permite explorar diferentes estrategias de resolución de laberintos, destacando el uso de programación orientada a objetos y lógica de autómatas.**

***Palabras clave*—Pygame, Generación de Laberintos, Agentes Autónomos, Python, Simulación.**

I. Introducción

El diseño y la simulación de laberintos han sido utilizados como herramientas educativas y de investigación en inteligencia artificial y programación. Este proyecto implementa un sistema interactivo donde agentes autónomos navegan a través de un laberinto generado proceduralmente para alcanzar un objetivo predefinido.

El sistema utiliza Python y Pygame para la generación visual y lógica de los laberintos, y emplea algoritmos diversos que definen los movimientos de los agentes en función de estrategias como búsqueda aleatoria, exploración sistemática o recorridos óptimos.

II. Estructura del Código

El código se divide en cuatro módulos principales, cada uno con responsabilidades específicas:

A. `maze_generator.py`

Este módulo genera laberintos de forma procedural utilizando un algoritmo basado en pilas y retroceso. Incluye funciones para identificar celdas disponibles, marcar posiciones visitadas y establecer un objetivo final en el laberinto.

B. `maze_app.py`

Este módulo maneja la interfaz gráfica del juego, configurando la ventana principal, los botones de interacción y el sistema de temporización. También controla el ciclo principal del juego y permite al usuario seleccionar entre tres modos de movimiento: `AgentT1`, `AgentT2` y `AgentT3`.

C. `entities.py`

Define las clases principales que representan a los agentes y su comportamiento:

- **MazeGenerator**: Clase base que incluye funciones para mover agentes y manipular el laberinto.
- **AgentT1**: Agente que explora de forma aleatoria, evitando revisitar posiciones previas consecutivas.
- **AgentT2**: Agente que prioriza evitar celdas previamente visitadas.
- **AgentT3**: Agente que implementa un enfoque más estructurado y analiza múltiples rutas para llegar al objetivo.

D. `cons.py`

Este módulo almacena las constantes del sistema, como los colores, dimensiones y configuraciones gráficas.

III. Modos de Movimiento

El comportamiento de los agentes varía según el modo seleccionado:

- **Aleatorio (`AgentT1`)**: Movimientos sin dirección específica, basados en valores generados al azar.
- **Profundidad (`AgentT2`)**: Navegación con registro de celdas visitadas para optimizar el recorrido.
- **Anchura (`AgentT3`)**: Estrategia de búsqueda informada que evalúa múltiples rutas hacia el objetivo.

IV. Interfaz

La interfaz del laberinto cuenta con un tablero en donde se proyecta el laberinto generado en cada ejecución. En la parte superior de la pantalla se muestra un cronometro que marca el tiempo de ejecución que lleva el agente recorriendo el laberinto. Del lado derecho se muestran tres botones. El primero botón **Inicio** ejecuta la acción de iniciar el recorrido del agente seleccionado en busca de la meta. El segundo botón **Pausa** detiene el movimiento del agente en el momento. El tercer botón permite seleccionar entre los tres modos de búsqueda (agentes) implementados. Finalmente se muestran dos botones extras que permiten modificar la velocidad del movimiento del agente en el laberinto. La interfaz del laberinto se muestra en la figura 1.

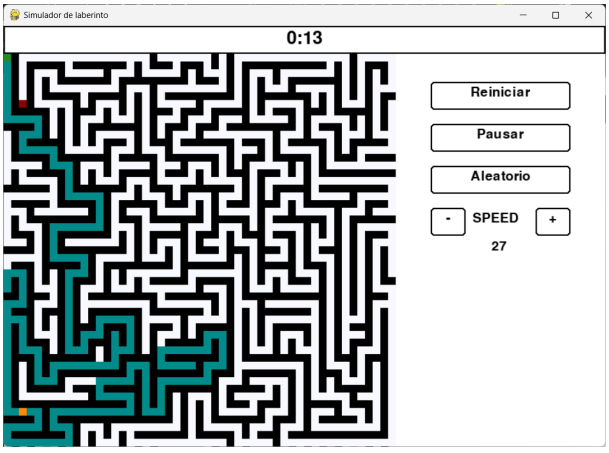

Fig. 1. Interfaz del laberinto. Se puede identificar el inicio del laberinto por el recuadro en color verde, la meta por el recuadro en color rojo y al agente como el recuadro de color naranja.

V. Tiempos de Ejecución

Se realizaron tres experimentos para medir la eficiencia de los métodos implementados en los diferentes agentes. Para esto se fijo una semilla en la función `maze_generator` para ejecutar los agentes en el mismo laberinto con las salidas y metas en las mismas posiciones.

Para el primer experimento se utilizó el método **aleatorio**. Este agente recorrió gran parte del laberinto analizando cerca del 100% de los pasillos. El agente llegó a la meta en un tiempo de ocho minutos con siete segundos (figura 2).

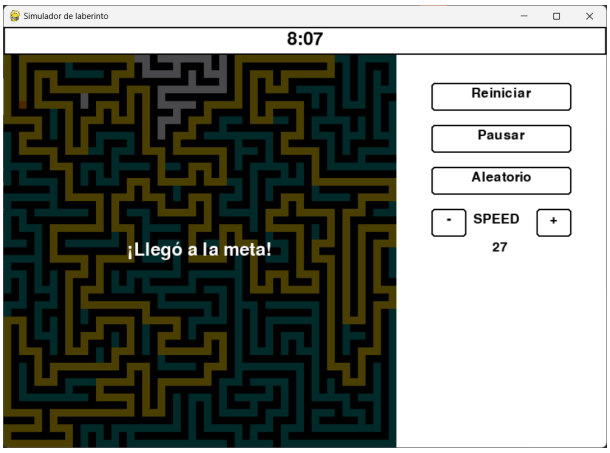

Fig. 2. Pantalla final del recorrido del agente con el método aleatorio. El camino correcto a la meta se muestra en color dorado. El agente llegó a la meta en un tiempo de ocho minutos con siete segundos.

Para el segundo experimento se utilizó el método de **profundidad**. Este agente, al igual que el primero, recorrió gran parte del laberinto analizando poco más del 50% de los pasillos. El agente llegó a la meta en un tiempo de cuarenta y cuatro segundos (figura 3).

Para el tercer y último experimento se utilizó el método de **anchura**. Este agente unicamente recorrido el pasillo del camino correcto hacia la meta. Para esto analizó cada una de sus opciones posibles en cada intersección que se encontraba para tomar la opción correcta. El agente llegó a la meta en un tiempo récord de veinticinco segundos (figura 4).

VI. Conclusiones

El proyecto destaca el uso de Python para combinar generación procedural de laberintos con lógica de agentes autónomos. La modularidad del diseño facilita la incorporación de nuevos modos

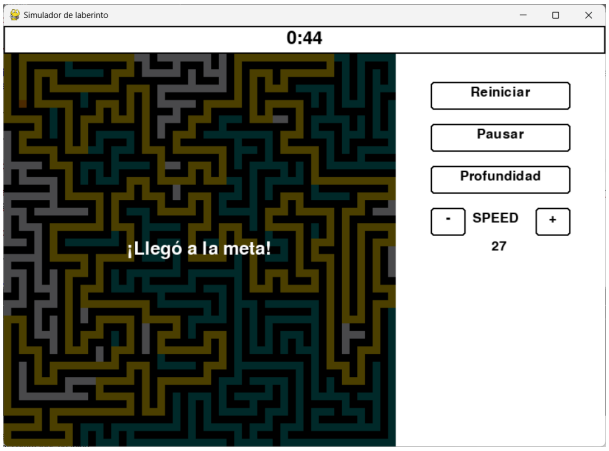

Fig. 3. Pantalla final del recorrido del agente con el método de profundidad. El camino correcto a la meta se muestra en color dorado. El agente llegó a la meta en un tiempo de cuarenta y cuatro segundos.

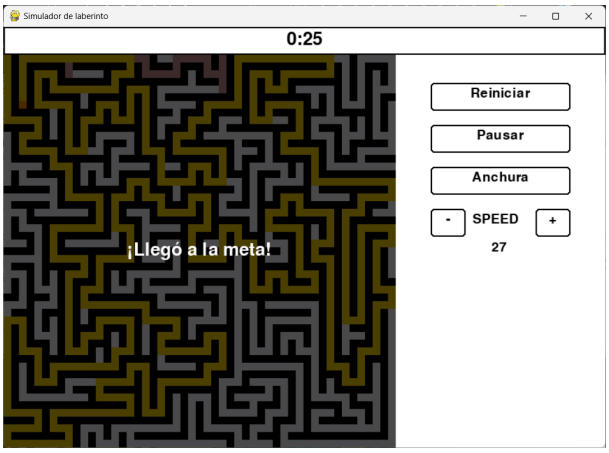

Fig. 4. Pantalla final del recorrido del agente con el método de anchura. El camino correcto a la meta se muestra en color dorado. Los caminos posibles analizados por el agente se muestran en rojo. El agente llegó a la meta en un tiempo de veinticinco segundos.

de movimiento y características adicionales, como la generación de estadísticas o el ajuste de la dificultad del laberinto. Este sistema proporciona una base sólida para proyectos educativos o de investigación en inteligencia artificial y simulaciones.

VII. Referencias

Pygame. (2024). *Pygame documentation*. Retrieved from https://www.pygame.org/docs/

Python Software Foundation. (2024). *Python documentation*. Retrieved from