PROYECTO 1: INTELIGENCIA ARTIFICIAL BÚSQUEDA NO INFORMADA

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL.

Presentado por:

Johan Steven Muñoz Lopez - 1958380
Jose Daniel Ospina Hincapie - 1958374
Sebastian Dario Giraldo Rodas - 2259391

Presentado a:

JOSHUA DAVID TRIANA MADRID

Universidad del Valle Sede Tuluá

Estudiantes de Ingeniería de Sistemas

TULUÁ - VALLE DEL CAUCA

2023-1

Tabla de contenido

1. **INTRODUCCIÓN.** **3**

1.1 Contextualización.

1.2 Definición del problema.

2. **OBJETIVO.** **4**

3. **DESARROLLO**.

3.1 Creacion de Personajes.

3.1.1 Pinocho.

3.1.2 Geppeto.

3.1.3 Cigarrillos.

3.1.4 Zorro.

4. IMPLEMENTACIÓN. **4**

4.1. Métodos usados. 4

4.2. Pruebas. 6

5. CONCLUSIONES. **8**

6. REFERENCIAS. **8**

1. INTRODUCCIÓN.

1.1 Contextualización: En este proyecto se aborda la exploración de una matriz mediante diferentes técnicas de búsqueda, entre las que se incluyen la búsqueda por amplitud, la búsqueda por costo y la búsqueda por profundidad iterativa. El objetivo principal es desarrollar un programa que sea capaz de aplicar estas técnicas para encontrar soluciones óptimas en la matriz, según los criterios definidos para cada tipo de búsqueda. Para lograr esto, se utilizará un enfoque de inteligencia artificial y se aplicarán los conceptos y algoritmos fundamentales de la teoría de búsqueda en grafos.

1.2 Definición del problema: El problema que se aborda en este proyecto es el de explorar una matriz mediante diferentes técnicas de búsqueda, con el fin de encontrar soluciones óptimas según los criterios establecidos para cada tipo de búsqueda.

2. OBJETIVO.

El objetivo es construir un programa que resuelva el problema de exploración de la matriz utilizando técnicas de búsqueda no informada, ya mencionados anteriormente y conceptos de inteligencia artificial.

3. DESARROLLO

3.1 Creación de personajes: Para este proyecto, tendremos 4 personajes.

3.1.1 *Pinocho*: Este personaje es el que explorará la matriz hasta encontrar a Geppeto, como características tiene que se puede mover hacía arriba, abajo, derecha e izquierda y en la matriz es representado por el valor 4

3.1.2 *Geppeto*: Este personaje será la meta, al cual pinocho tendrá que llegar, este personaje no tiene ningún movimiento en la matriz.

3.1.3 *Cigarrillos*: Este no es un personaje en sí, más bien es un obstáculo, el cual si pinocho pasa por éste dentro de la matriz, tendrá un costo de 2 por movimiento.

3.1.4 *Zorro*: Este no es un personaje en sí, más bien es un obstáculo, el cual si pinocho pasa por éste dentro de la matriz, tendrá un costo de 3 por movimiento.

4. IMPLEMENTACIÓN.

Para la implementación se usó Python, la interfaz fue realizada con la librería pygame.

4.1 MÉTODOS USADOS.

El proyecto se divide en 4 clases principales:

- ***juego.py***

  Esta es la clase principal, esta utiliza la biblioteca Pygame para crear una ventana de juego en la que se puede cargar una matriz desde un archivo de texto y realizar diferentes tipos de búsqueda no informada (búsqueda por costo, búsqueda por amplitud, búsqueda por profundidad iterativa) en la matriz.

  La estructura del código consiste en tres funciones principales:

  load_matrix es una función que carga una matriz de un archivo de texto y la devuelve como una lista de tuplas.

  load_images es una función que carga imágenes y las almacena en un diccionario.

  draw_matrix es una función que dibuja la matriz y las imágenes en la pantalla de juego.

  Después de definir estas tres funciones, el código crea una ventana principal utilizando la biblioteca Pygame y muestra un menú con opciones para realizar diferentes tipos de búsqueda. Cuando se hace clic en una opción, el código cambia el título de la ventana y llama a la función draw_matrix con la matriz cargada y la ruta de búsqueda correspondiente.

- ***amplitud.py***

  En esta clase se implementa una función llamada "bfs" que realiza una búsqueda en anchura para encontrar el camino más corto desde una posición inicial hasta una meta en una matriz dada.

  La estructura de datos que utiliza para almacenar los nodos a visitar es una cola (deque) y también utiliza un conjunto (set) para almacenar las posiciones ya visitadas. Además, utiliza un diccionario para almacenar los costos de los nodos visitados.

- ***profundidad.py***

  En esta clase se implementa una función llamada “dfs” que realiza una búsqueda en profundidad iterativa con un límite de profundidad de 10 (se establece un límite de 10 para evitar que el algoritmo se ejecute indefinidamente si no se encuentra la solución).

Este utiliza una pila y una lista para almacenar los nodos visitados y una variable para almacenar los costos de cada posición. La función itera a través de los nodos en la pila y los vecinos de cada nodo para explorar el camino más corto a la meta.

- ***costo.py***

  En esta clase se implementa la función "*costo_uniforme*" el algoritmo de búsqueda de costo uniforme se usa para encontrar el camino más corto entre un nodo de inicio y un nodo objetivo en una matriz representando un laberinto.

  Este utiliza una cola de prioridad donde cada nodo es una tupla que contiene el costo del camino hasta el nodo, la posición del nodo y el camino hasta el nodo desde el nodo de inicio. Además se implementa un diccionario de costos, que asigna un costo a cada tipo de casilla en la matriz. Si una casilla no está en el diccionario, se asume que su costo es 1.

4.2 PRUEBAS.

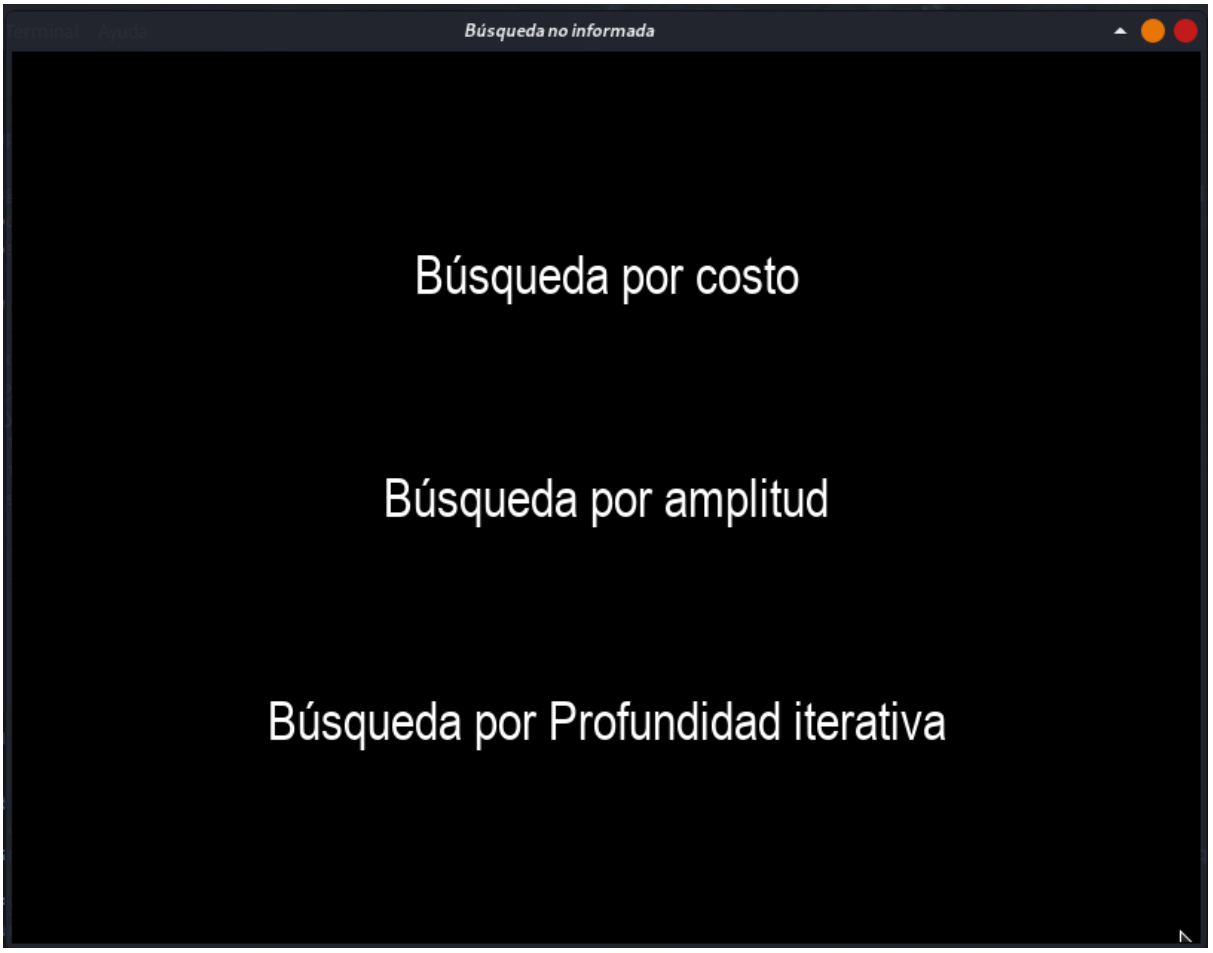

Imagen 1. Prueba del menú.

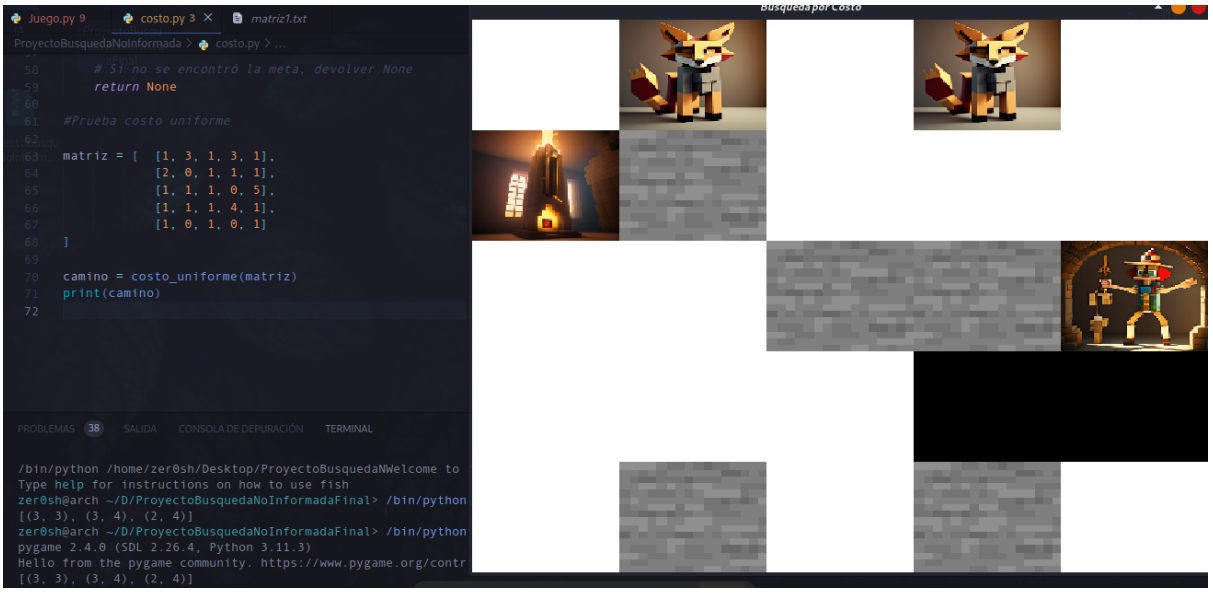

Imagen 2. Prueba unitaria del método de costo. junto con su respectiva representación gráfica.

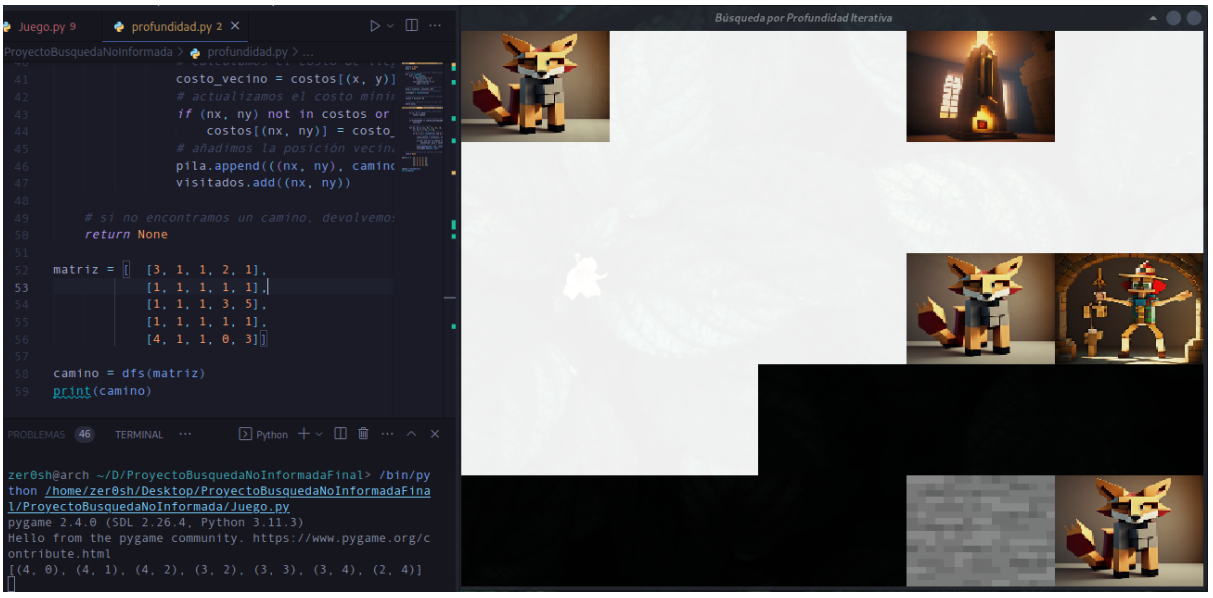

Imagen 3. Prueba unitaria del método de recorrido por profundidad. junto con su respectiva representación gráfica.

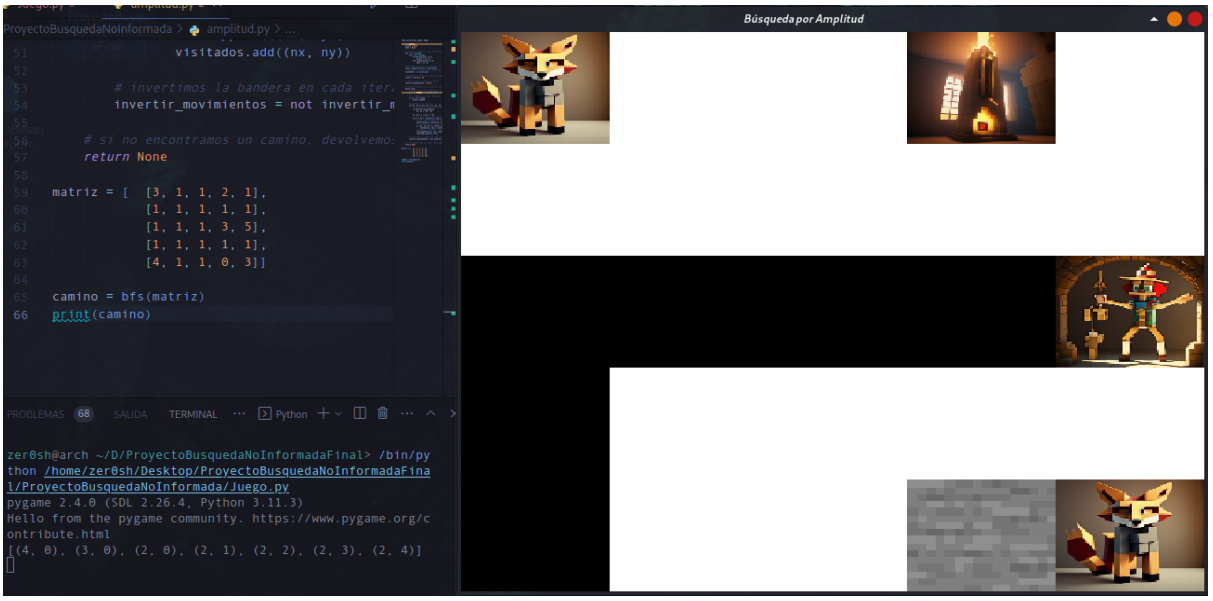

Imagen 4. Prueba unitaria del método de recorrido por Amplitud. junto con su respectiva representación gráfica

5. CONCLUSIONES.

Con base en la implementación de recorridos de búsqueda no informada con Python y Pygame, se logró desarrollar un sistema que permite encontrar la solución óptima a problemas de búsqueda en un espacio de estados. El proyecto incluyó la implementación de los algoritmos de búsqueda en anchura, búsqueda en profundidad, búsqueda en profundidad limitada y búsqueda de costo uniforme, así como la visualización del proceso de búsqueda en tiempo real utilizando la librería Pygame.

La búsqueda en amplitud (BFS) es útil cuando se busca una solución óptima en un espacio de búsqueda finito y no pesado, ya que siempre encuentra la solución más cercana a la raíz, pero puede ser ineficiente en espacios de búsqueda infinitos o pesados, debido a que puede requerir demasiada memoria para almacenar la frontera de búsqueda.

La profundidad iterativa puede ser más eficiente en espacios de búsqueda profundos o infinitos, ya que realiza la búsqueda de manera incremental, sin almacenar toda la frontera de búsqueda de una vez. Sin embargo, puede requerir una cantidad significativa de tiempo en espacios de búsqueda con profundidad desconocida, por ello fijamos la profundidad en 10.

En general, ambos algoritmos son útiles y eficientes en diferentes situaciones. La elección del algoritmo a utilizar dependerá de las características del problema y del espacio de búsqueda en cuestión.

6. REFERENCIAS.

- OpenAI. (2021). GPT-3.5. OpenAI. https://openai.com/
- Caparrini, F. S., & Windmill Web Work. (s/f). *Fernando Sancho caparrini*. Cs.Us.Es. Recuperado el 6 de mayo de 2023, de http://www.cs.us.es/~fsancho/
- Franco, H. (s/f). *Búsqueda no informada*. Universidad Central. Recuperado el 6 de mayo de 2023, de https://hpclab.ucentral.edu.co/~hfranco/int_art/Session_6_UninformedSearch.pdf
- Millán, M. (2011). *Comparación de métodos de búsqueda.*