INFORME PROYECTO 1

BUSQUEDAS NO INFORMADAS

INTEGRANTES:

HAROLD ANDRÉS MONTANO HURTADO (1968067-3743)

JUAN STEBAN VELOZA GUEVARA (1968025-3743)

UNIVERSIDAD DEL VALLE

INGENIERÍA DE SISTEMAS

INTELIGENCIA ARTIFICIAL

TULUÁ-VALLE

2023

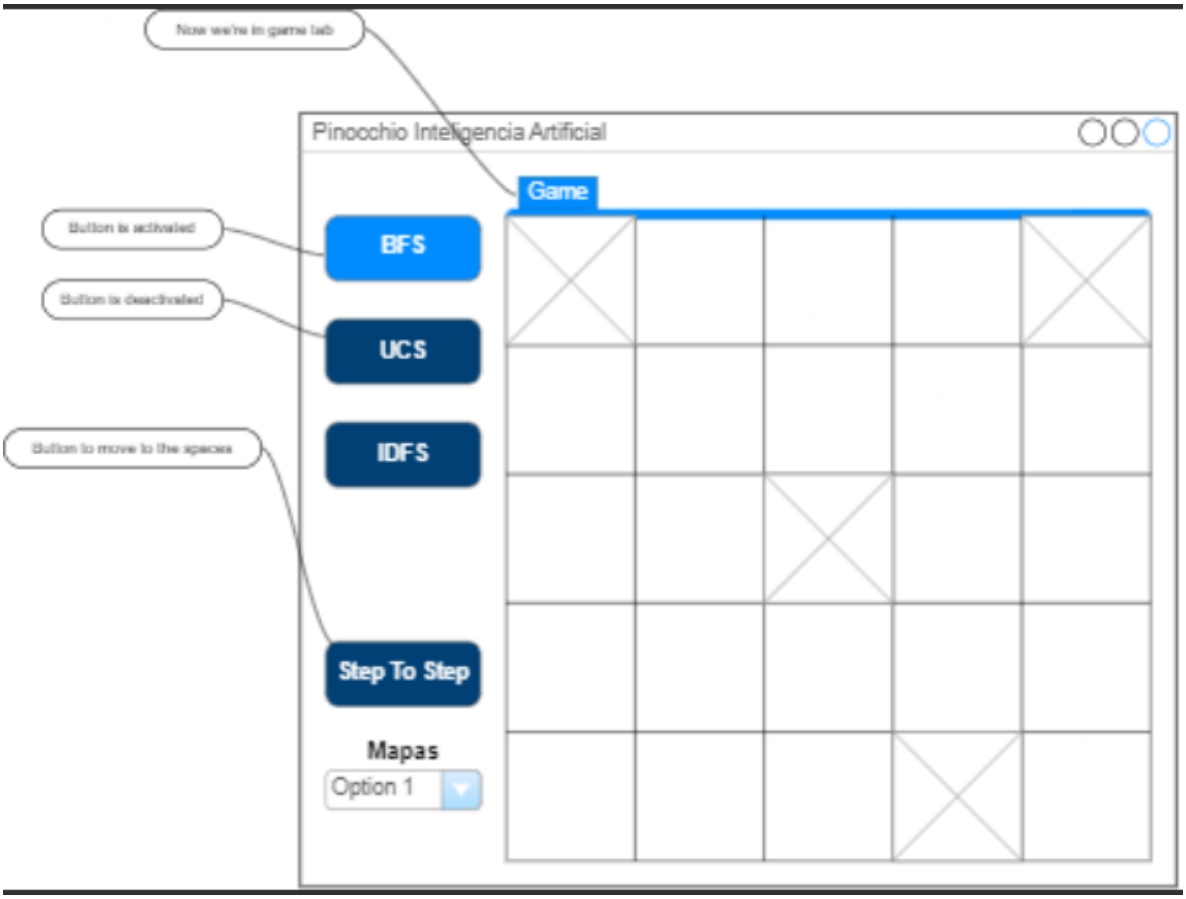
CONTENIDO

INTRODUCCIÓN

Teniendo en cuenta que la inteligencia artificial ha tenido un acelerado crecimiento en el uso de las tecnologías modernas, donde gracias a esto, el uso de esta novedosa tecnología se ha ido implementando poco a poco en el desarrollo de aplicaciones que han ido revolucionando la industria de la tecnología. Asociándolo con este proyecto se tiene como fin, aplicar el conocimiento adquirido, para otorgarle a un agente (entidad capaz de percibir su entorno, procesar estas percepciones y actuar en su entorno de manera correcta) cierto aprendizaje para que éste pueda desarrollar una acción, si es el caso fallar y saber cómo remediar dicho error. En el caso de este proyecto se le equipará de ciertas estructuras de aprendizaje algorítmicos, por medio del cual el agente pueda resolver algún tipo de problema que se le presente a la hora de ir recorriendo cada paso del laberinto hasta poder llegar a su objetivo final. Para el proyecto, se hace que el laberinto sea recorrido por el personaje animado “Pinocho” el cual, debe pasar ciertos obstáculos los cuales tienen cierto costo de pasar por allí, para poder llegar a donde” Gepetto”. En los siguientes pasos, se podrá ver el análisis requerido para que este proyecto sea óptimo y cumpla con las expectativas requeridas.

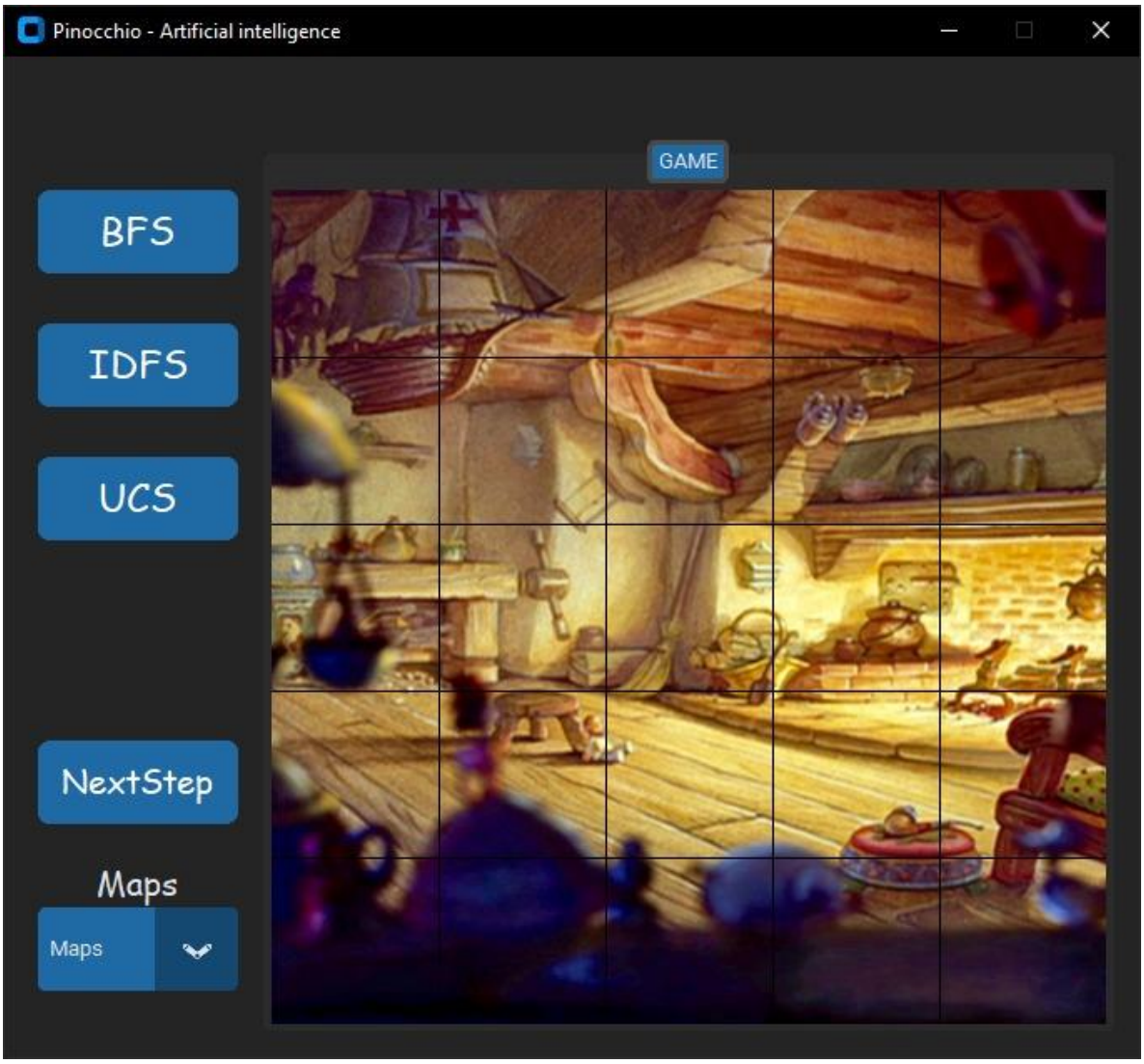
SISTEMA PROPUESTO

Mediante el siguiente modelo se planea tener una idea clara de las opciones que va a tener la aplicación que tiene como fin desarrollar desenvolverse en un ambiente, el cual tiene varios obstáculos por donde pasará el agente (Pinocho), dichos obstáculos tienen algún tipo de precio pasar por ellos. Es clave aclarar, que el sistema propuesto pretende que el agente, los obstáculos y la meta tengan diversos tipos de ubicaciones, es decir, que no haya una ubicación fija, lo que si será fijo es el tamaño de la matriz (5x5) donde actuará este ambiente. Las siguientes imágenes son los modelos que se tienen planeados para la aplicación final, una idea global a donde se pretende llegar con el fin de tener un rumbo fijo de diseño.



Este es el diseño a implementar propuesto, el cual va a ir con el siguiente entorno el cual fue desarrollado cada imagen por la inteligencia artificial de

Microsoft (bing), a continuación, se anexan las imágenes que han sido añadidas dentro de la aplicación.


Pinocchio - Artificial intelligence
GAME
BFS
IDFS
UCS
NextStep
Maps
Maps

IMPLEMENTACIÓN

- Para la implementación del algoritmo de búsqueda por amplitud que se realizó en este proyecto se usaron las siguientes estructuras para desarrollar dicho algoritmo:
  - Conjunto (set), el cual es utilizado para almacenar los nodos que ya fueron visitados, para que no se visiten nuevamente.
  - Matriz, se usa para representar el grafo a revisar.
  - La cola, esta estructura busca mantener una lista de nodos que se buscan procesar. Esta estructura se usa aprovechando su construcción de tipo FIFO (first input, first output).

  Teniendo en cuenta las anteriores estructuras, se crean como argumentos de la función de búsqueda por amplitud, la matriz a recibir que representa el laberinto a recorrer. Comienza creando una cola para almacenar los nodos que se deben visitar. Luego, se crea un nodo para la posición inicial en la matriz y se agrega a la cola de nodos por visitar. La función utiliza un conjunto para almacenar los nodos ya visitados lo que ayuda a evitar ciclos infinitos. Posteriormente, se ingresa en un bucle while que continúa visitando nodos hasta que la cola este vacía.

  Dentro del bucle, se obtiene el siguiente nodo de la cola y se verifica si este nodo es el objetivo. Llegando a ser el caso de que sea el objetivo, se crea el camino hasta dicho nodo y se devuelve en un orden invertido, porque el camino que se construyó del nodo final al nodo inicial. Si el nodo actual no es el nodo objetivo, se obtienen sus vecinos. Siguiente, estos nodos vecinos son agregados a la cola de nodos por visitar si no han sido visitados anteriormente. Si se da el caso de que ya hayan sido visitados, no se agregan a la cola evitando ciclos infinitos.

  De lo contrario si no se encuentra un camino hasta el objetivo, se devuelve “None”. Esta función utiliza una cola para almacenar los nodos por visitar y un conjunto para evitar ciclos infinitos; después visita en orden el camino y construye el camino más corto.

- Para la implementación del algoritmo de costo uniforme que se implementó en este proyecto se usaron las siguientes estructuras para desarrollar dicho algoritmo:
    - Heap, en este caso es utilizada como una cola de prioridad, en la cual los elementos se organizan de tal manera que el elemento de mayor prioridad esté siempre en la cabeza de la cola. En un heap o montículo, los elementos se organizan con forma de árbol binario de búsqueda completo, y cada nodo tiene un valor menor o igual que los valores de sus hijos. Gracias a esta librería que dispone Python, se permite agregar o extraer elementos del montículo.
    - Diccionario, el cual es usado para guardar los nodos con sus respectivos valores.

  Se inicia la función de búsqueda por costo uniforme, creando un nodo de inicio y agregándolo a la cola de prioridad. A continuación, se crea un diccionario para rastrear los nodos que ya fueron explorados y sus costos; la cola de prioridad es un heap o montículo, el cual se ordena automáticamente por el costo de cada nodo, esta ejecución se hace efectiva hasta que la cola de prioridad esté vacía o se encuentre la meta. En cada recorrido del ciclo, el nodo con el costo más bajo se extrae de la cola de prioridad. Si es el nodo meta, la función reconstruye la ruta desde el nodo meta hasta el inicio. Si este no es el objetivo se obtienen los vecinos y se agregan a la cola, si estos no se han explorado, también son agregados al diccionario con sus debidos costos.

- Para la implementación del algoritmo de búsqueda por profundidad iterativa que se implementó en este proyecto se usaron las siguientes estructuras para desarrollar dicho algoritmo:
    - La pila, esta estructura busca almacenar los nodos a explorar, en este caso, se hace uso de un array para representar la entrada por matriz. Esta estructura se usa aprovechando su construcción de tipo LIFO (last input, first output).

  La idea en la implementación de este algoritmo es buscar el objetivo explorando gradualmente cada vez más profundo en el árbol de

búsqueda, pero sin expandir los nodos a la vez. Se inicializa la pila con la raíz del árbol de búsqueda. Teniendo en cuenta cada iteración que genera el bucle, se extrae el último elemento de la pila, que es el último nodo visitado, comprobando si es el nodo meta. Llegando a ser así, se devuelve la ruta que llevó hasta el nodo. Si no se ha alcanzado la profundidad máxima, se empiezan a generar los hijos del nodo actualmente siendo visitado y se añaden a la pila junto con la ruta que llevó hasta el nodo actual; si se ha alcanzado la profundidad máxima, se continua con el siguiente elemento de la pila. Si no se encuentra el objetivo, se devuelve None.

PLAN DE PRUEBAS

El plan de pruebas que se dará a conocer que se realizó en la implementación de cada algoritmo, a continuación, se mostrará cada implementación realizada para cada algoritmo de búsquedas:

- **Búsqueda por amplitud:**

1. Prueba de matriz sin obstáculos:

   Entrada: matriz = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15], [16,17,18,19,20], [21,22,23,24,25]]

   Salida: matriz = [(0, 0), (0,1), (0,2), (0,3), (0,4), (1,4), (2,4), (3,4), (4,4)]

2. Prueba de matriz con obstáculos:
   Entrada: matriz = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15], [16,17,18,19,20], [21,22,23,24,25]]
   Salida: matriz = None
3. Prueba de matriz con múltiples elementos y su solución:
   Entrada: matriz = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15], [16,17,18,19,20], [21,22,23,24,25]]
   Salida: matriz = [(0, 0), (0,1), (0,2), (1,2), (2,2), (3,2), (4,2), (4,3), (4,4)]

- **Búsqueda por costo uniforme:**

1. Matriz con un único camino posible desde la posición (0,0) hasta la posición (4,4) con un costo total de 8.

   Entrada: matriz = [[2,1,1,1,1], [1,1,1,1,1], [1,1,1,1,1], [1,1,1,1,1], [1,1,1,1,2]]

   Inicio = (0,0)

   Meta = (4,4)

2. Matriz con varios caminos posibles desde la posición (0,0) hasta la posición (4,4) pero el camino más corto tiene un costo total de 12.

Entrada: matriz = [[2,1,1,1,1], [1,1,1,1,1], [1,2,2,2,1], [1,1,1,1,1], [1,1,1,1,2]]

Inicio = (0,0)

Meta = (4,4)

3. Matriz con obstáculos que obligan al algoritmo a buscar un camino alternativo con mayor costo.

   Entrada: matriz = [[2,1,1,1,1], [1,0,1,1,1], [1,2,2,2,1], [1,1,0,1,1], [1,1,1,1,2]]

   Inicio = (0,0)

   Meta = (4,4)

- **Búsqueda por profundidad iterativa:**

1. Prueba de caso base con matriz inicial igual a la matriz objetivo.
   Matriz inicial: [[1,2,3,4,5], [6,7,8,9,1], [2,3,4,5,6], [7,8,9,1,2], [3,4,5,6,7]]
   Matriz meta: [[1,2,3,4,5], [6,7,8,9,1], [2,3,4,5,6], [7,8,9,1,2], [3,4,5,6,7]]
   Profundidad máxima: 10
   Resultado: Ruta completa desde la matriz inicial hasta la matriz objetivo.

2. Prueba de caso complejo con matriz inicial aleatoria a la matriz objetivo posible después de varias iteraciones.
   Matriz inicial: [[1,2,3,4,5], [6,7,8,9,1], [2,3,4,5,6], [7,8,9,1,2], [3,4,5,6,7]]
   Matriz meta: [[1,2,3,4,5], [6,7,8,9,1], [2,3,4,5,6], [7,8,9,1,3], [3,4,5,6,7]]
   Profundidad máxima: 5
   Resultado: Ruta desde la matriz inicial hasta una matriz que se acerca a la matriz objetivo dentro de la profundidad máxima establecida.
3. Prueba de caso fallo con matriz inicial aleatoria, matriz objetivo imposible de alcanzar.
   Matriz inicial: [[1,2,3,4,5], [6,7,8,9,1], [2,3,4,5,6], [7,8,9,1,2], [3,4,5,6,7]]
   Matriz meta: [[1,2,3,4,5], [6,7,8,9,1], [2,3,4,5,6], [7,8,9,1,3], [3,4,5,6,7]]
   Profundidad máxima: 2

Resultado: None, porque la matriz objetivo no es posible percibirse dentro de la profundidad máxima establecida.

CONCLUSIONES

Para finalizar, se puede deducir que gracias a estos algoritmos de búsqueda es posible capacitar a un sistema para que pueda resolver ciertos problemas, también se puede ayudar a que un agente pueda tener errores, pero lo particular es que gracias a estos algoritmos pueda aprender a como resolver esos errores y buscar un camino que permita resolver esos conflictos, estos algoritmos de búsqueda son proporcionados gracias a los árboles de búsqueda por medio de los cuales se permiten resolver mucho problemas de la computación.