**Dibujo en blanco y negro

Descripción generada automáticamente con confianza bajaINSTITUTO POLITÉCNICO NACIONAL**

ESCUELA SUPERIOR DE CÓMPUTO

**NOMBRES:**

PADILLA MATIAS CRISTIAN MICHEL

SAUCILLO GONZÁLEZ JESSE OBED

**GRUPO:**

4BM1

**TRABAJO:**

Practica 2

"Búsqueda no informada"

**MATERIA:**

Fundamentos de Inteligencia Artificial

**FECHA**

25 de septiembre del 2023

Logotipo

Descripción generada automáticamente

**PRACTICA 1. AGENTES REACTIVOS**

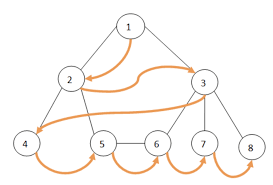
**RESUMEN**

En esta práctica se realizo la creación de una aplicación de un 15-puzzle; un tablero con una imagen segmentada en 15 partes y 1 espacio en blanco de manera desordenada, donde la meta será intercambiar el espacio en blanco por cada segmento fronterizo hasta dar con el orden correcto. Dentro de esta aplicación se desarrollaron los algoritmos de búsqueda primero en anchura y búsqueda primero en profundidad, estos pueden ser seleccionados por el usuario para resolver el 15-puzzle.

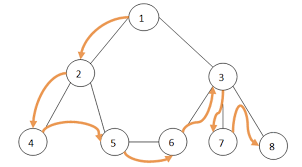
**INTRODUCCIÓN**

La búsqueda no informada, a veces llamada búsqueda ciega o búsqueda a ciegas, es un enfoque de búsqueda en la inteligencia artificial donde el algoritmo no tiene información previa sobre la ubicación o la estructura del objetivo que está tratando de encontrar. En otras palabras, no utiliza ninguna heurística o conocimiento adicional para guiar la búsqueda hacia la solución. En lugar de eso, se basa únicamente en la exploración sistemática del espacio de búsqueda, existen 2 estrategias de búsqueda no informada: búsqueda primero en anchura y búsqueda primero en profundidad.

**Búsqueda primero en anchura (BFS -** **Breadth First Search)**

La estrategia de búsqueda en anchura comienza por explorar el nodo raíz y luego se expanden gradualmente los nodos sucesores, seguidos de sus sucesores, y así sucesivamente. Se exploran todos los nodos en el mismo nivel de profundidad en el árbol de búsqueda antes de pasar al siguiente nivel de profundidad.

**Búsqueda primero en profundidad (DFS –****Depth First Search)**

En la estrategia de búsqueda en profundidad, se inicia desde el nodo raíz y se sigue una de las ramificaciones del árbol lo más lejos posible hasta encontrar el nodo deseado o alcanzar un nodo hoja, que es un nodo sin descendientes. Si se llega a un nodo hoja, se continúa la búsqueda ascendiendo hacia el ancestro más cercano que tenga nodos hijos sin explorar.

**15-Puzzle**

El 15-puzzle es un juego de rompecabezas que consiste en una cuadrícula de 4x4 con 15 fichas numeradas y un espacio vacío. El objetivo del juego es mover las fichas para ordenarlas en orden numérico, dejando el espacio vacío en la esquina inferior derecha. Para resolver el 15-puzzle, se pueden utilizar diferentes algoritmos de búsqueda, como la búsqueda primero en anchura (BFS) y la búsqueda primero en profundidad (DFS).

**DESARROLLO**

El código de la práctica se realizó en lenguaje Java. Para el desarrollo de la aplicación se modifico el proyecto “Pozole” proporcionado por el profesor. Este proyecto contiene las clases: “Pozole”, “Tablero” “State”, y “Executor”. A continuación, se explica a detalle cada clase.

**Pozole**

Esta es la clase principal que contiene el método main. En este método se crea una instancia de la clase Tablero y se muestra en pantalla.

Interfaz de usuario gráfica, Texto, Aplicación, Teams

Descripción generada automáticamente

**Tablero**

Esta es la clase principal del juego y extiende la clase JFrame de Java Swing para crear una ventana de juego. Aquí se configura la interfaz gráfica del juego, se carga la imagen del rompecabezas, se manejan eventos de botones y se inicia la resolución del rompecabezas utilizando los algoritmos de búsqueda en anchura (BFS) o búsqueda en profundidad (DFS) según la elección del usuario.

initComponents: Configura la interfaz gráfica del juego, crea y muestra botones, y maneja eventos.

Texto, Aplicación

Descripción generada automáticamente

goodBye: Cierra la aplicación.

Texto

Descripción generada automáticamente con confianza baja

gestionarExit: Maneja la acción de salida de la aplicación.

Texto

Descripción generada automáticamente con confianza media

imagePieces: Divide una imagen en piezas para usar en el rompecabezas.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

paintPieces: Coloca las imágenes en los botones del tablero de juego. Texto

Descripción generada automáticamente

whichMethod: Determina si se utilizará BFS o DFS para resolver el rompecabezas. Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamentesolve: Inicia la resolución del rompecabezas usando BFS o DFS.

Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

muestraEstados: Muestra información sobre el estado actual durante la búsqueda.

Texto

Descripción generada automáticamente con confianza media

repetido: Comprueba si un estado ya se ha visitado para evitar bucles infinitos.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media**State**

Esta clase representa un estado del rompecabezas y contiene métodos relacionados con la manipulación de los estados, la generación de estados sucesores y la verificación del estado objetivo. State: El constructor inicializa un estado a partir de una cadena que representa la disposición actual del rompecabezas.

show: Muestra el estado del rompecabezas en la consola. Tabla

Descripción generada automáticamente

swap: Intercambia dos fichas en el estado.

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamentenextStates: Genera estados sucesores posibles a partir del estado actual.

getBoard: Obtiene el tablero de juego actual.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

goalFunction: Verifica si el estado actual es igual al estado objetivo.

Texto

Descripción generada automáticamente

getI y getJ: Obtiene las coordenadas de la ficha vacía en el tablero.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamenteImagen que contiene Diagrama

Descripción generada automáticamente

getMovement: Obtiene el último movimiento realizado en el estado. Imagen que contiene Texto

Descripción generada automáticamente

isEqual: Compara dos estados para ver si son iguales. Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente**Executor**

Esta clase se utiliza para ejecutar la animación de los movimientos del rompecabezas después de encontrar la solución. Se encarga de mover las fichas del rompecabezas de acuerdo con la solución encontrada.

**Comparación del desempeño de los algoritmos**

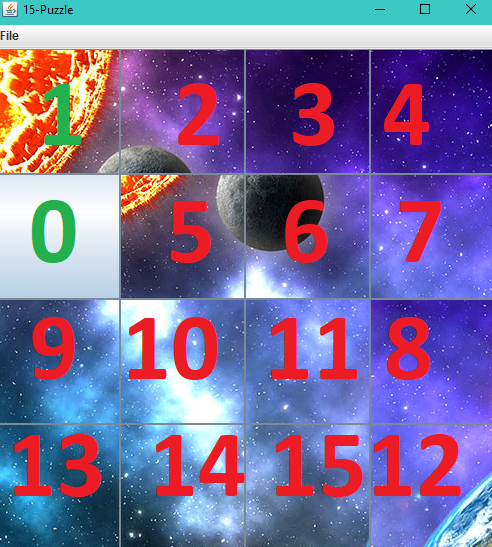
Caso de estudio

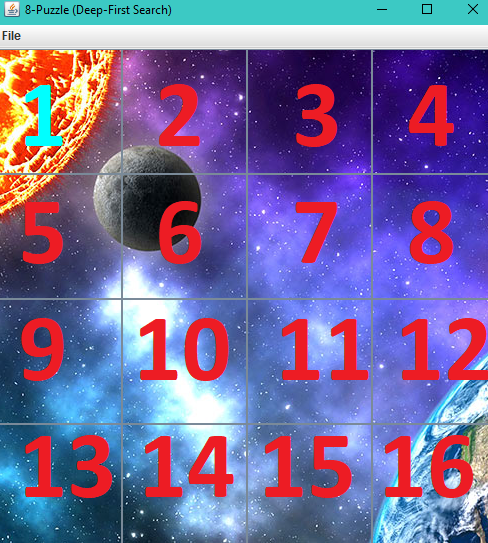
Se ha establecido una configuración inicial de las piezas del puzle con la variable “start” de la clase Tablero. Esta configuración se resolverá con Breadth First Search y Depth First Search.

Configuración start en el código:

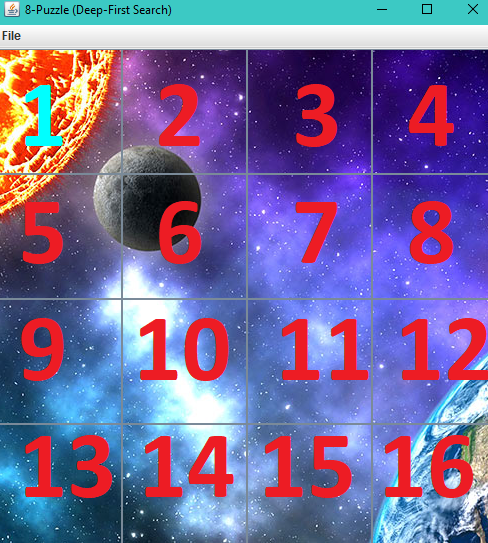


Configuración inicial gráficamente:



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteBreadth First Search Resultado:

Depth First Search Resultado:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La configuración "123405679AB8DEFC", es un caso interesante para analizar por qué en esta ocasión la Búsqueda en Profundidad (DFS) fue más eficiente que la Búsqueda en Anchura (BFS).

Profundizando en el análisis tenemos que:

La BFS tiende a explorar estados más profundos antes de volver atrás. En este caso, la profundidad máxima alcanzada fue de 176 nodos entre todos los niveles.

DFS puede explorar profundamente en un camino antes de explorar otros. En este caso, la profundidad máxima alcanzada fue de 17 nodos entre los niveles explorados.

La configuración inicial fue más favorable para DFS. Ambos algoritmos tienen lógica para evitar estados repetidos y callejones sin salida. Sin embargo, en este caso, DFS encontró caminos sin salida menos complejos y manejo mejor los estados repetidos. Mientras que para BFS la aleatoriedad en la cola (debido a la exploración de varios caminos en el mismo nivel) afecto negativamente la eficiencia.

**CONCLUSIONES**  
El juego del 15-puzzle es un desafío soluble utilizando algoritmos de búsqueda no informada, como la búsqueda primero en anchura (BFS) y la búsqueda primero en profundidad (DFS). Ambos algoritmos ofrecen enfoques diferentes para abordar el problema y tienen sus propias ventajas y desventajas.

BFS es eficiente para encontrar la solución más corta en términos de movimientos, esto es importante cuando se busca una solución óptima en términos de la menor cantidad de movimientos posible. DFS, por otro lado, se sumerge profundamente en un camino antes de retroceder, lo que puede ser más eficiente en términos de memoria y puede encontrar soluciones en un tiempo razonable.

Ambos algoritmos de búsqueda no informada son valiosos en la resolución de problemas de búsqueda y proporcionan una base sólida para abordar una variedad de desafíos. La elección del algoritmo adecuado dependerá de las características específicas del problema (como el tamaño del tablero, la configuración inicial) y de si se prioriza encontrar la solución óptima en términos de movimientos o se busca una solución rápida y eficiente en términos de recursos.**BIBLIOGRAFÍA**

Lukegarrigan. (2021, 5 Junio). *What is Breadth-First Search (BFS) - Codeheir*. Codeheir. https://codeheir.com/2021/06/05/what-is-breadth-first-search-bfs/

Alfonso López. (2020, 11 mayo). *Algoritmo primero el mejor (BFS) C++* [Vídeo]. YouTube. https://www.youtube.com/watch?v=MQ8ZKqE4S2c

‌