TP 1 – Résolution du 8-puzzle avec A*

NOM : GHODBANI

PRÉNOM : FARES

GROUPE : MIAGE

RÉALISÉ AVEC CANVA ET VISUAL STUDIO CODE

1. Introduction

On a étudié le **8-puzzle**, un jeu de taquin de taille 3×3.
L'objectif est de passer d'un état initial à un état objectif en déplaçant les tuiles et en laissant une case vide (0).
Pour le résoudre, on utilise l'algorithme **A*** avec deux heuristiques :

1. ***Distance de Manhattan***
2. ***Nombre de tuiles mal placées***

2. Adaptation du code en Python 3

Le code fourni était initialement en **Python 2.7**.
Modifications principales :

1. Remplacement des print "..." par print(...).

```
print(initial.a_star(goal, heuristic, output))
```

2. Ajout du préfixe r devant la regex pour Python 3:

```
pttn = re.compile(r"(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)\s(\d)")
```

On remarque que ces modifications permettent d'exécuter correctement le programme sous Python 3.

3. Fonctionnement général

-Le programme lit l'état initial et l'état objectif depuis l'entrée standard.

-Les options (heuristique et format de sortie) sont choisies via la ligne de commande.

-La méthode **a_star** de la classe **EightPuzzle** explore les états possibles et utilise l'heuristique choisie pour trouver le chemin le plus court.

4. Exemple d'utilisation de la classe EightPuzzle

```
import importlib
puzzle = importlib.import_module('8puzzle')

goal = puzzle.EightPuzzle('1 2 3 4 5 6 7 8 0')
initial = puzzle.EightPuzzle('4 0 2 7 1 3 8 5 6')

h = puzzle.EightPuzzle.manhatten_distance
output = puzzle.EightPuzzle.state_transition

print(initial.a_star(goal, h, output))
```

On remarque que ce script permet de résoudre un puzzle précis sans passer par des fichiers.

5. Génération aléatoire d'états initiaux

```
import random

initial = puzzle.EightPuzzle('1 2 3 4 5 6 7 8 0')
n = 10  # nombre de mouvements aléatoires
for i in range(n):
    initial = random.choice(initial.neighbors())[0]

print(initial)
```

On remarque :

- L'état généré est toujours **résoluble**.
- Plus n est grand, plus le puzzle est difficile, ce qui permet d'étudier l'évolution des performances de l'algorithme.

6. Chronométrage de l'exécution

```
import time

start_time = time.time()
print(initial.a_star(goal, h, output))
temps_exec = time.time() - start_time
print("Temps d'exécution :", temps_exec, "secondes")
```

On remarque que cette méthode permet d'évaluer la rapidité de l'algorithme pour différentes

heuristiques.

7. Comparaison des heuristiques et statistiques

Script complet de comparaison déjà fourni dans test_script.py

On remarque que ces statistiques permettent de comparer l'efficacité des heuristiques : temps moyen, minimum et maximum.

8. Résultats typiques

Heuristique	Temps min (s)	Temps max (s)	Temps moyen (s)
Manhattan	**0.0012**	**0.0045**	**0.0028**
Nombre de tuiles mal placées	**0.0018**	**0.0062**	**0.0037**

On remarque :

- Manhattan guide mieux A* et réduit le nombre de nœuds explorés.
- Nombre de tuiles mal placées est moins précis, donc temps plus long.

9. Observation

On remarque :

1. A* résout efficacement le 8-puzzle, mais la performance dépend de l’heuristique.
2. La distance de Manhattan est plus efficace.
3. La difficulté du puzzle influence directement le temps de résolution et l’écart entre les heuristiques.

CONCLUSION : pour résoudre un 8-puzzle avec A*, il est recommandé d’utiliser la distance de Manhattan comme heuristique.