**Présentation du projet de puzzle 8 en Python avec Tkinter**

**Slide 1 :**

**Introduction**

Le projet consiste à implémenter le jeu 8-puzzle en utilisant l'interface graphique Tkinter de Python, en développant des algorithmes d'intelligence artificielle pour résoudre le puzzle automatiquement. Le jeu se joue sur une grille 3x3 avec huit tuiles numérotées de 1 à 8 et une case vide. L'objectif est de réorganiser les tuiles pour atteindre un état final spécifique.

**Slide 2:**

**Algorithmes de recherche:** Le projet implémente trois algorithmes de recherche différents pour résoudre le puzzle :

* + A\* : Algorithme de recherche Optimal qui combine le coût parcouru et une estimation heuristique pour trouver la solution optimale.
* BFS (Best-First Search) : Algorithme de recherche informée qui évalue l'opportunité d'un nœud par rapport à un autre en utilisant une heuristique, effectuée sur l'ensemble des nœuds dans la liste OUVERT.

**Slide 3 :**

Structure de developpement avec python :

Les Classes pythons :

1. Classe EightPuzzle:

Cette classe représente le jeu du 8-puzzle ,c’est la classe main qui contient la logique de l'interface utilisateur ainsi que les interactions avec les algorithmes de résolution.

1. Classe AStarSolver

Cette classe implémente l'algorithme A\* pour résoudre le puzzle.

1. Classe BFSSolver

Cette classe implémente l'algorithme de recherche en largeur (BFS) pour résoudre le puzzle.

**Slide 4 :**

Les Bibliotheque python utilisés

* Tkinter : Utilisé pour la création de l'interface graphique.
* PriorityQueue : Utilisé dans l'algorithme A\* pour gérer la frontière de recherche.
* heapq : Utilisé dans l'algorithme BFS pour gérer la file de priorité.
* deque : Utilisé dans l'algorithme Hill Climbing pour gérer la file d'attente des nœuds à explorer.
* random : Utilisé pour mélanger les tuiles lors de l'initialisation du puzzle.

**Slide 5 (ken lezem 2 slide hot w baad taw nchoufou):**

Fonction Heuristique (h(n))

Distance de Manhattan

A computer screen shot of a program

Description automatically generated

* Somme des distances absolues entre les positions actuelles et les positions cibles des tuiles.
* Calculée comme :

A black background with white text

Description automatically generated

* Simple à calculer et efficace pour les puzzles de glissement.

Exemple :

A screenshot of a cellphone

Description automatically generatedEtat courant : Etat final :

A screenshot of a game

Description automatically generated

distance de manhatan : h(n) = 2 + 2 + 0 + 2 + 2 + 0 + 3 + 3 = 14

**Slide 6**

A screenshot of a cellphone

Description automatically generatedl’etat Initial : L’etat final :

A screenshot of a number game

Description automatically generated

**Slide 7:**

A Star

A\* utilise une file d'attente prioritaire pour stocker les états potentiels à explorer. Chaque état est associé à une valeur f(x) calculée comme la somme de son coût parcouru (g(x)) et de son coût estimé restant (h(x)). L'algorithme explore les états suivants :

La fonction g(x) correspond au nombre de mouvements effectués pour atteindre l'état actuel depuis l'état initial, pour chaque voisin, il calcule le nouveau coût parcouru (g(x)) en ajoutant 1 au coût parcouru de l'état parent.

**Slide 8**

Exemple dimplementation : (7ot capture video w m3aha jomltin hedhom louta)

Résolu en 21 étapes

Résolu en 1829 itérations

**Slide 9**

Best-First Search (BFS)

Pour le Best-First Search (BFS), l'algorithme utilise une approche similaire à A\* mais sans tenir compte du coût parcouru jusqu'à présent. Au lieu de cela, il évalue chaque état uniquement en fonction de son estimation du coût restant pour atteindre l'objectif, ce qui est représenté par la fonction heuristique h(x).

**Slide 10**

Exemple dimplementation : (7ot capture video bfs w m3aha jomltin hedhom louta)

Résolu en 47 étapes

Résolu en 89 itérations

**Slide 11**

**Conclusion**

En conclusion, l'algorithme A\* a généralement produit des solutions optimales avec le nombre minimum de mouvements, mais il peut nécessiter un nombre important d'itérations, surtout pour des états initiaux complexes.

La meilleure recherche d'abord (BFS) peut être rapide mais peut conduire à des solutions sous-optimales où elle a nécessité plus de mouvements que les autres algorithmes. Le choix entre A\* et BFS dépend donc des ressources computationnelles disponibles et du niveau d'optimalité souhaité pour le chemin de solution.

**Slide 12 :**

**Algorithme A**\* :

* Produit des solutions optimales avec le nombre minimum de mouvements, mais peut nécessiter un nombre important d'itérations, surtout pour des états initiaux complexes.

**Recherche en Largeur (BFS)** :

* Peut être plus rapide, mais peut conduire à des solutions sous-optimales nécessitant plus de mouvements que les autres algorithmes.

Mise en place d'une interface utilisateur intuitive avec Tkinter.

Intégration réussie des algorithmes A\* et BFS pour la résolution du puzzle.

Application des algorithmes de recherche A et BFS

Animation des mouvements et des transitions entre les états pour une meilleure visualisation des solutions.

Comparaison de l'efficacité des algorithmes en termes de vitesse et de nombre de mouvements nécessaires.