



PROJET IA

8 PUZZLES GAME

Ranim Abdellatif

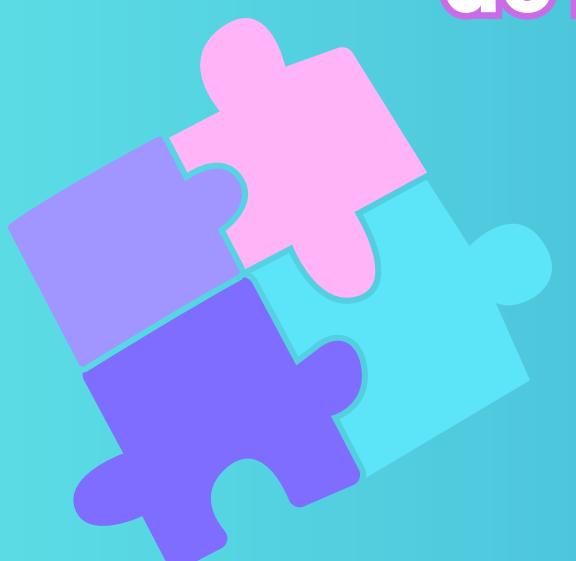
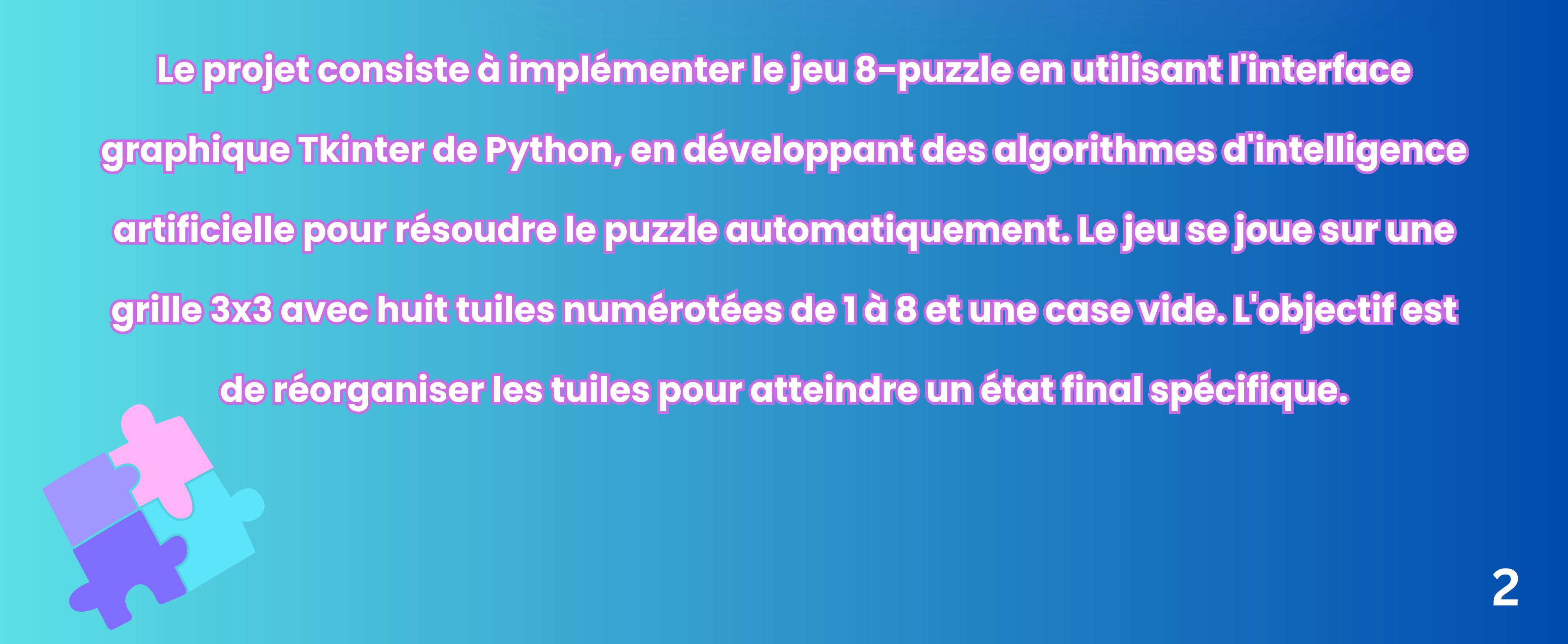
Rihem Amri

Ranim Satouri



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.



ALGORITHMES DE RECHERCHE

Le projet implémente deux 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.

STRUCTURE DE DEVELOPPEMENT AVEC PYTHON

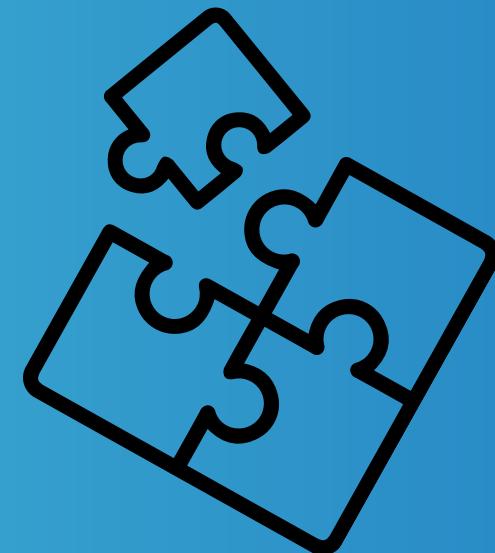
Les Classes pythons :

- **Classe Eight Puzzle:** 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.
- **Classe A* Solver:** Cette classe implémente l'algorithme A* pour résoudre le puzzle.
- **Classe BFS Solver:** Cette classe implémente l'algorithme de recherche en largeur (BFS) pour résoudre le puzzle.

LES BIBLIOTHEQUE PYTHON UTILISÉS

Les Bibliothèque 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.
- **random**: Utilisé pour mélanger les tuiles lors de l'initialisation du puzzle.



Fonction Heuristique ($h(n)$)

Distance de Manhattan

```
1 @staticmethod
2 def heuristic(state, goal_state):
3     # Manhattan distance heuristic
4     distance = 0
5     for i in range(3):
6         for j in range(3):
7             if state[i][j] != goal_state[i][j] and state[i][j] != 0:
8                 x, y = divmod(goal_state[i][j], 3)
9                 distance += abs(x - i) + abs(y - j)
10    return distance
```

Somme des distances absolues entre les positions actuelles et les positions cibles des tuiles.

- Calculée comme :

$$h(n) = \sum_{i=0}^2 \sum_{j=0}^2 |x_i - x_{goal}| + |y_i - y_{goal}|$$

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



etat courant

distance de manhattan :

$$h(n) = 2 + 2 + 0 + 2 + 2 + 0 + 3 + 3 = 14$$

etat final



2	7	3
8	5	1
6	4	

ETAT INITIAL

1	2	3
4	5	6
7	8	

ETAT FINAL

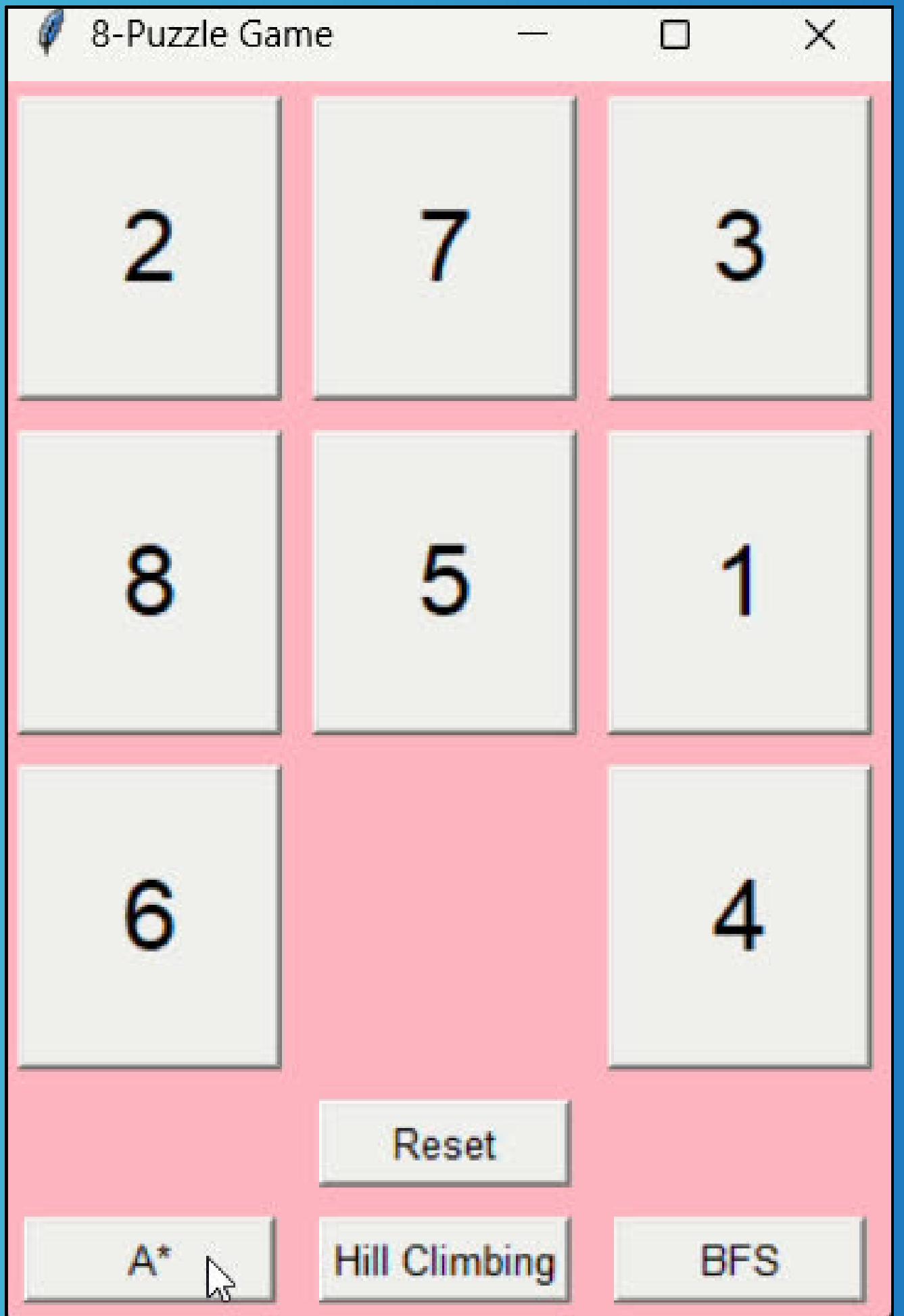
EXEMPLE A*

- 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 ($h(x)$).
- 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.



EXEMPLE D'IMPLEMENTATION

Résolu en 21 étapes
Résolu en 1829 itérations



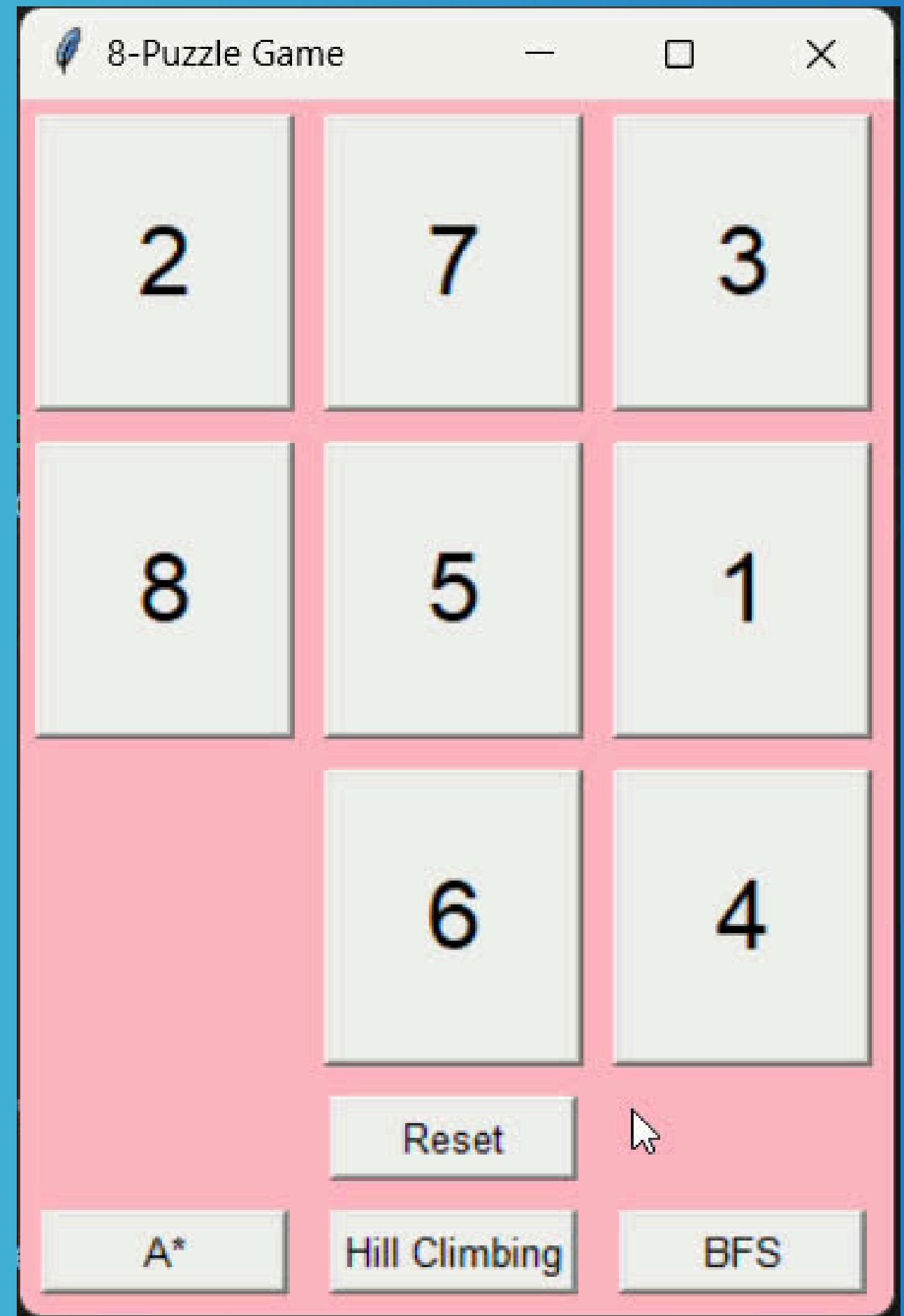
BEST-FIRST SEARCH (BFS) :

- 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 $h(x)$.



EXAMPLE DIMPLEMENTATION

Résolu en 47 étapes
Résolu en 89 itérations



COMPARAISON ENTRE LES ALGORITHME

Algorithme A* :

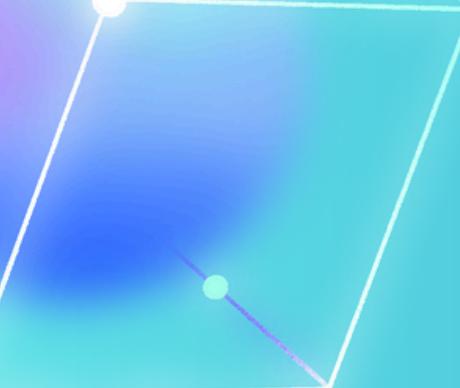
- Produit des solutions optimales avec le nombre minimum de mouvements, mais peut nécessiter un nombre important d'itérations, surtout pour des états initiaux complexes.

Best First Search (BFS) :

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

CONCLUSION

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



MERCI POUR VOTRE ATTENTION

