Rapport recherche sur le Jeu Taquin

1. Introduction

Ce projet a été réalisé seul durant le 1er semestre du master informatique de l'université d'artois, dans le cadre de la matière Introduction à l'Intelligence Artificielle, dans le but d'appliqué des algorithmes de recherche tels que Astar, DFS et BFS sur un jeu.

Cela a été réalisé en python (c'etait une obligation).

Ce rapport présente l'analyse des performances de différents algorithmes de recherche appliqués au jeu du Taquin. Le jeu du Taquin est un puzzle avec une case vide déplaçable composé de (k x k -1) case numérotées dans un cadre k x k, où k représente la dimension du plateau. L'objectif est de réorganiser les case dans l'ordre en déplaçant la case vide.

2. Implémentation

2.1 Structure du Projet

Le projet est structuré autour de plusieurs composants clés :

- Une classe JeuTaquin gérant la logique du jeu
- Des implémentations des algorithmes de recherche (BFS, DFS, A*)
- Un système de tests automatisés complet

2.2 Choix d'Implémentation

1. Représentation de l'État

- Utilisation d'un dictionnaire associant les valeurs aux positions
- La case vide est représentée par la valeur 0
- Cette structure permet un accès et une modification efficaces

2. Vérification de Résolvabilité

- Implémentation du théorème des inversions pour garantir que les configurations générées sont résolvables
- Adaptation selon la parité de la taille du plateau

3. Gestion des Mouvements

- Implémentation des quatre directions possibles (haut, bas, gauche, droite)
- Validation des mouvements selon la position de la case vide

2.3 Spécificités des Algorithmes

1. A* (astar.py)

- Implémentation d'une heuristique de distance de Manhattan
- Gestion dynamique de la mémoire avec système de branches
- Limite de temps paramétrable pour éviter les calculs infinis
- Stockage optionnel du chemin de solution

2. BFS (bfs.py)

- Utilisation d'une structure FilePile personnalisée
- Exploration systématique niveau par niveau
- Gestion efficace des états visités
- Stockage optionnel du chemin de solution

3. DFS (dfs.py)

- Limitation de profondeur paramétrable
- Exploration prioritaire en profondeur
- Optimisation de la mémoire
- Stockage optionnel du chemin de solution

2.4 Structure de Données FilePile

La classe FilePile implémente une structure de données hybride avec les caractéristiques suivantes:

- Double fonctionnalité (file/pile) selon le mode d'insertion
- Implémentation par liste doublement chaînée
- Gestion automatique de la mémoire

J'ai repris la structure d'une class nommé FilePile que j'avais réalisé en Licence, je l'ai modifié un petit peu pour qu'elle soit plus adapté et optimisé.

3. Tests et Résultats

3.1 Information

Seulement A* est testé sur une grille de taille 4 puisque dfs et bfs ne sont pas en mesure de résoudre le jeu taquin sur ce genre de taille en raison de la compléxité trop élevé pour ce genre de taille.

Les tests ont été effectué sur un pc avec cette configuration:

-Windows 11 23h

-i7 13700

-32go de ram

-AMD RX 7800XT

3.2 Résultats des tests - 2024-11-24 13:53:

Résultats pour grille 2 x 2

ASTAR

- Tests complétés: 1000
- Tests échoués: 0
- Temps minimum: 0,000004053116 secondes
- Temps maximum: 0,000282526016 secondes
- Temps moyen: 0,000022450209 secondes

BFS

- Tests complétés: 1000
- Tests échoués: 0
- Temps minimum: 0,000004291534 secondes
- Temps maximum: 0,000225543976 secondes
- Temps moyen: 0,000034480095 secondes

DFS

- Tests complétés: 1000
- Tests échoués: 0
- Temps minimum: 0,000003814697 secondes
- Temps maximum: 0,000300084247 secondes
- Temps moyen: 0,000033607968 secondes

Résultats pour grille 3 x 3

ASTAR

- Tests complétés: 1000
- Tests échoués: 0
- Temps minimum: 0,000102996826 secondes
- Temps maximum: 0,069900142098 secondes
- Temps moyen: 0,007481940746 secondes

BFS

- Tests complétés: 1000
- Tests échoués: 0
- Temps minimum: 0,001555442810 secondes
- Temps maximum: 1,602700948715 secondes
- Temps moyen: 0.834813608217 secondes

DFS

- Tests complétés: 1000
- Tests échoués: 0
- Temps minimum: 0.000989675522 secondes
- Temps maximum: 1.544354915619 secondes
- Temps moyen: 0.626475614071 secondes

Résultats pour grille 4 x 4

ASTAR

- Tests complétés: 762
- Tests échoués: 238
- Temps minimum: 0.004545211792 secondes
- Temps maximum: 40.760850667953 secondes
- Temps moyen: 9.883096008163 secondes

4. Analyse Comparative des Algorithmes

4.1 Points Forts et Limitations

A*

- **Points Forts** :
    - Trouve toujours le chemin optimal
    - Performances excellentes sur les petites grilles
    - Gestion efficace de la mémoire grâce au système de branches
- **Limitations** :
    - Consommation mémoire importante sur les grandes grilles
    - Peut échouer sur le 4×4 dans les cas complexes

BFS

- **Points Forts** :
    - Garantie de trouver la solution optimale
    - Implémentation simple et robuste
    - Excellent sur les petites grilles
- **Limitations** :
    - Explosion exponentielle de la mémoire
    - Inadapté aux grilles 4×4 et plus
    - Temps de calcul important sur les grilles moyennes

DFS

- **Points Forts** :
    - Consommation mémoire linéaire
    - Peut trouver rapidement une solution
    - Adapté à l'exploration profonde
- **Limitations** :
    - Ne garantit pas l'optimalité
    - Risque de boucles infinies sans limite de profondeur
    - Performances variables selon les cas

5. Recommandations et Perspectives d'Amélioration

5.1 Optimisations possible

1. Optimisation de l'Heuristique A*

- Améliorer l'heuristique

2. Techniques d'Élagage

- Détection rapide des branches non prometteuses
- Réduction dynamique du facteur de branchement
- Mémorisation des sous-problèmes résolus

5.2 Plan d'Implémentation

Phase 1 : Optimisations Immédiates

- Amélioration de la gestion mémoire
- Implémentation des techniques d'élagage basiques

Phase 2 : Améliorations Avancées

- Parallélisation des calculs

6. Stockage et affichage du chemin de résolution

6.1 Introduction

Les algorithmes de recherche (A*, BFS, DFS) implémentés pour résoudre le jeu de Taquin peuvent stocker et afficher le chemin complet de la solution. Cette fonctionnalité permet de visualiser pas à pas comment l'algorithme a résolu le puzzle, ce qui est particulièrement utile pour comprendre et comparer les différentes stratégies de résolution. Il est nécessaire de lancer le fichier **main.py** pour pourvoir afficher les chemins ou faire une partie.

Toutefois les résultats des tests ont été lancée via le fichier **test.py** pour pouvoir tester sur un grands nombre de grilles différente.

Implémentation

Paramètre de stockage du chemin

Chaque algorithme de recherche (A*, BFS, DFS) accepte un paramètre booléen optionnel `stocker_chemin` qui, lorsqu'il est défini à `True`, active le stockage du chemin de résolution :

```
def astar(jeu, etat_initial, etat_final, stocker_chemin=False):
# ...

def bfs(jeu, etat_initial, etat_final, stocker_chemin=False):
# ...

def dfs(jeu, etat_initial, etat_final, stocker_chemin=False):
# ...
```

Fonctionnement interne

Lorsque `stocker_chemin` est activé :

1. L'algorithme maintient un dictionnaire `parents` qui associe chaque état à son état parent
2. Quand la solution est trouvée, la fonction `reconstruire_chemin()` est appelée pour reconstituer le chemin complet
3. Le chemin est stocké dans l'attribut `solution_path` de l'instance `JeuTaquin`

Format du chemin stocké

Le chemin est stocké sous forme d'une liste d'états, où chaque état est représenté par sa clé unique (chaîne de caractères). Cette liste commence par l'état initial et se termine par l'état final.

Impact sur les performances

Il est important de noter que le stockage du chemin de résolution du Jeu du Taquin a un impact sur :

- L'utilisation mémoire : stockage supplémentaire pour le dictionnaire des parents
- Le temps d'exécution : léger surcoût pour maintenir les relations parent-enfant

Pour cette raison, cette fonctionnalité est optionnelle et peut être désactivée si seule la solution finale est nécessaire.

7. Conclusion

On remarque que l'algorithme A* est de loin bien plus performant que bfs et dfs. Par contre bfs et dfs sont très proche et sont tout deux incapable de fonctionné sur une grille de taille 4.