L'Algorithme A* (A-Star)

L'algorithme A* est un algorithme de recherche de chemin très populaire en informatique et en intelligence artificielle. Son objectif principal est de trouver le chemin le plus court entre un point de départ et un point d'arrivée, en évitant les obstacles et en minimisant le coût (par exemple, la distance ou le temps).

Dans notre jeu de taquin, un "chemin" est une séquence de mouvements de tuiles, et le "coût" est simplement le nombre de mouvements. A* est utilisé par l'IA pour trouver la solution la plus rapide au puzzle.

La Formule Clé : `f(n) = g(n) + h(n)`

Le génie de A* réside dans cette simple formule qui lui permet de décider quel chemin explorer en priorité.

- `n` : Représente un "nœud", qui dans notre cas est une configuration spécifique du plateau de jeu (un état du puzzle).
- `g(n)` **(Coût du chemin)** : C'est le coût **réel** pour aller du point de départ jusqu'au nœud `n`. Pour le puzzle, c'est tout simplement le **nombre de mouvements déjà effectués** pour arriver à la configuration `n`.
- `h(n)` **(Heuristique)** : C'est une **estimation intelligente** du coût pour aller du nœud `n` jusqu'à la solution. C'est une "supposition éclairée". Pour que A* fonctionne correctement, cette estimation ne doit **jamais surestimer** le coût réel. Dans notre jeu, on utilise la distance de Manhattan.
- `f(n)` **(Score total)** : C'est le score total estimé du chemin passant par le nœud `n`. A* va toujours choisir d'explorer en premier le nœud qui a le plus petit score `f(n)`.

Comment ça marche, étape par étape ?

Imaginez que l'algorithme a deux listes :

1. **L'Open Set** : Une liste de toutes les configurations de puzzle qu'il a découvertes mais qu'il n'a pas encore explorées. C'est sa "liste de choses à faire". Elle est triée par le score `f(n)` le plus bas.
2. **Le Closed Set** : Une liste de toutes les configurations qu'il a déjà visitées et analysées. C'est pour ne pas refaire le même travail.

Voici le processus :

1. **Initialisation** : L'algorithme commence par mettre la configuration initiale (le puzzle mélangé) dans l'Open Set.
2. **Boucle de recherche** : Tant qu'il y a des configurations dans l'Open Set :
   a. Il prend la configuration avec le **plus petit score** `f(n)` dans l'Open Set. Appelons-la `current`.

b. Si `current` est la solution (le puzzle est résolu), alors c'est gagné ! L'algorithme a trouvé le chemin le plus court.
c. Sinon, il déplace `current` de l'Open Set vers le Closed Set.
d. Il génère toutes les configurations possibles à partir de `current` en un seul mouvement (ses "voisins").
e. Pour chaque voisin : - S'il est déjà dans le Closed Set, on l'ignore. - Sinon, on calcule son score `f(n)`. S'il n'est pas déjà dans l'Open Set ou si ce nouveau chemin pour y arriver est meilleur, on l'ajoute ou on le met à jour dans l'Open Set.

3. **Fin** : Si l'Open Set devient vide et que la solution n'a pas été trouvée, cela signifie qu'il n'y a pas de solution possible.

Application à votre Jeu de Taquin

- **État/Nœud (`n`)** : C'est la liste `puzzle.state`, qui représente la position de chaque tuile.
- **Coût du chemin (`g(n)`)** : C'est le nombre de mouvements, `len(path)`. Chaque mouvement ajoute +1 au coût.
- **Heuristique (`h(n)`)** : La **Distance de Manhattan**.

Focus sur la Distance de Manhattan

La distance de Manhattan est nommée ainsi car elle représente la distance qu'un taxi devrait parcourir dans une ville quadrillée comme Manhattan.

La formule est : `distance = |x1 - x2| + |y1 - y2|`

- `(x1, y1)` sont les coordonnées actuelles de la tuile.
- `(x2, y2)` sont les coordonnées où la tuile devrait être dans la solution.

L'heuristique `h(n)` totale pour une configuration de puzzle est la **somme des distances de Manhattan de toutes les tuiles mal placées**.

Exemple Détaillé du Déroulement

Prenons un puzzle 2x2 très simple. La case vide est représentée par le numéro 3.

État de départ : `[1, 3, 0, 2]` | **État final (solution) :** `[0, 1, 2, 3]`

Étape 0 : Initialisation

1. L'algorithme analyse l'état de départ `[1, 3, 0, 2]`.
2. Il calcule `g(n)` et `h(n)` :
   - `g(n) = 0` (on n'a encore fait aucun mouvement).
   - `h(n)` (distance de Manhattan totale) :
     - Tuile 0 : est en `(0,1)`, devrait être en `(0,0)`. Distance = `|0-0|+|1-0| = 1`.
     - Tuile 1 : est en `(0,0)`, devrait être en `(1,0)`. Distance = `|0-1|+|0-0| = 1`.
     - Tuile 2 : est en `(1,1)`, devrait être en `(0,1)`. Distance = `|1-0|+|1-1| = 1`.

        - **Total h(n) = 1 + 1 + 1 = 3**.
    - `f(n) = g(n) + h(n) = 0 + 3 = 3`.
3. Il ajoute cet état à l'Open Set.
    - **Open Set :** `[ (état=[1,3,0,2], f=3) ]`
    - **Closed Set :** `[ ]`

Étape 1 : Première Itération

1. A* prend l'état avec le meilleur score (le plus bas) de l'Open Set. Il n'y en a qu'un : `(état=[1,3,0,2], f=3)`.
2. Ce n'est pas la solution. Il le déplace dans le Closed Set.
3. Il génère tous les mouvements possibles à partir de cet état. La case vide (3) peut échanger avec la tuile 1 ou la tuile 2.
    - **Voisin A :** `[3, 1, 0, 2]` (échange avec la tuile 1)
        - `g(n) = 1` (1 mouvement depuis le départ).
        - `h(n)` = (calcul pour tuiles 0,1,2) = `1 + 0 + 1 = 2`.
        - `f(n) = 1 + 2 = 3`.
    - **Voisin B :** `[1, 2, 0, 3]` (échange avec la tuile 2)
        - `g(n) = 1`.
        - `h(n)` = (calcul pour tuiles 0,1,2) = `1 + 1 + 0 = 2`.
        - `f(n) = 1 + 2 = 3`.
4. Il ajoute les deux voisins à l'Open Set.
- **Open Set :** `[ (état=A, f=3), (état=B, f=3) ]`
- **Closed Set :** `[ (état=[1,3,0,2]) ]`

Étape 2 : Deuxième Itération

1. A* regarder l'Open Set. Les deux états ont le même score `f=3`. Il en choisit un (disons le A).
2. Il déplace A `[3, 1, 0, 2]` dans le Closed Set.
3. Il génère les voisins de A. La case vide (3) peut échanger avec la tuile 1 ou la tuile 0.
    - **Voisin C :** `[1, 3, 0, 2]` (échange avec 1). Cet état est déjà dans le Closed Set, donc on l'ignore.
    - **Voisin D :** `[0, 1, 3, 2]` (échange avec 0)
        - `g(n) = 2` (2 mouvements depuis le départ).
        - `h(n)` = (calcul...) = `0 + 0 + 1 = 1`.
        - `f(n) = 2 + 1 = 3`.
4. Il ajoute le Voisin D à l'Open Set.
- **Open Set :** `[ (état=B, f=3), (état=D, f=3) ]`
- **Closed Set :** `[ (état=[1,3,0,2]), (état=A) ]`

L'algorithme continue ce processus, en choisissant toujours l'état avec le plus petit `f(n)`, jusqu'à ce qu'il sorte de l'Open Set l'état final `[0, 1, 2, 3]`. À ce moment, il s'arrête et remonte le chemin qu'il a pris pour y arriver.

[Image de la visualisation de l'algorithme A*]

Conclusion

A* est efficace car il n'explore pas les chemins au hasard. Grâce à sa fonction heuristique `h(n)`, il explore de manière "intelligente" les chemins qui semblent les plus prometteurs pour atteindre rapidement la solution, ce qui lui permet de résoudre des puzzles complexes sans avoir à tester toutes les combinaisons possibles.