République Tunisienne
Ministère de l'Enseignement Supérieur, de la Recherche Scientifique et des Technologies de l'Information et de la Communication
Université de Carthage
Institut Supérieur des Technologies de l'Information et de la Communication

Travaux Pratiques
Fondements en Intelligence Artificielle (IA)

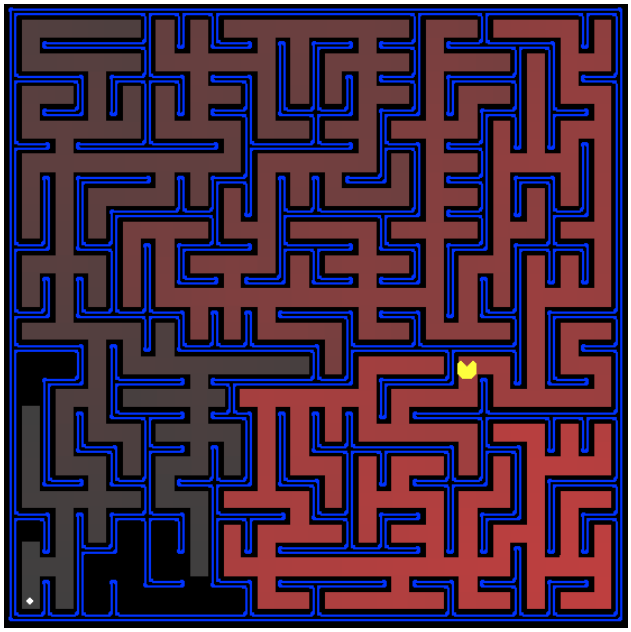
Enseignants : **Khaled Belghith, Manel Mrabet, Akram Khemiri**
Contenu élaboré par : **Khaled Belghith**
Filière / Niveau : **Informatique Fondamentale, Sciences de l'informatique ,2ème année**
Section / Groupe : **GLSI 2**

TP 1 – Algorithmes de recherche dans Pac-Man

Plan :

1. Introduction :

- Ce travail doit fonctionner avec la version 2.6 ou 2.7 du langage Python.



Pour ce travail, votre agent Pac-Man devra trouver des chemins dans des labyrinthes, que ce soit pour atteindre un endroit précis ou pour manger des gommes de façon efficace. Vous devrez implémenter

des algorithmes de recherche génériques et les appliquer à des scénarios dans le jeu de Pac-Man.

Dans les fichiers fournis pour ce travail, un correcteur automatique est inclus pour évaluer votre implémentation. Celui-ci peut être lancé par la commande :

```
~ $ python autograder.py
```

Le code de ce projet consiste en plusieurs fichiers Python. Seuls les fichiers **search.py** et **searchAgent.py** devront être modifiés et remis ; les autres fichiers peuvent être consultés au besoin, mais ne doivent pas être modifiés. De plus, seules les sections indiquées dans les fichiers **search.py** et **searchAgent.py** doivent être modifiées ; les noms des classes et des variables globales ainsi que les signatures des méthodes et des fonctions ne doivent pas être modifiées.

***Note** : Tous les algorithmes de recherche doivent retourner une liste d'actions qui mèneront l'agent de son état initial jusqu'à son but. Ces actions doivent être légales (e.g., direction valide, pas de déplacement au travers des murs*).

***Note** : Utilisez les structures de données Stack, Queue et PriorityQueue fournies dans le fichier utils.py pour implémenter vos algorithmes.*

2. Premiers pas

Après avoir téléchargé et extrait le code source, vous devriez pouvoir jouer à Pac-Man en lançant cette commande

```
~ $ python pacman.py
```

Le fichier **searchAgents.py** contient les agents intelligents utilisant des algorithmes de recherche, implémentés dans le fichier search.py. L'agent le plus simple est implémenté dans la classe GoWestAgent, qui se déplace toujours vers l'ouest :

```
~ $ python pacman.py --layout testMaze --pacman GoWestAgent
```

Bien entendu, cet agent trivial n'arrive pas à résoudre des labyrinthes plus complexes :

```
~ $ python pacman.py --layout tinyMaze --pacman GoWestAgent
```

Dans le fichier **searchAgents.py**, vous trouverez un agent complètement implémenté dans la classe **SearchAgent**, qui planifie un chemin dans un labyrinthe de Pac-Man et qui exécute ce chemin étape par étape. Votre travail consiste à implémenter les algorithmes de recherche qui seront utilisés par cet agent. Testez d'abord que le SearchAgent fonctionne correctement avant la commande :

```
~ $ python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

La commande ci-dessus indique que le SearchAgent doit utiliser la fonction tinyMazeSearch comme algorithme de recherche, implémenté dans search.py. Pac-Man devrait réussir à atteindre son objectif dans le labyrinthe.

3. Recherche en profondeur (depth-first search)

Implémentez un algorithme de recherche en profondeur dans la fonction depthFirstSearch du fichier **search.py**. L'algorithme doit être implémenté comme une recherche dans un graphe qui évite d'explorer les états déjà visités. Le code devrait rapidement trouver une solution pour les exemples ci-dessous :

```
~ $ python pacman.py -l tinyMaze -p SearchAgent

~ $ python pacman.py -l mediumMaze -p SearchAgent
```

Indice : *Si vous utilisez la* Stack *comme structure de données, la solution trouvée par l'algorithme de recherche en profondeur pour le labyrinthe* mediumMaze *devrait avoir une longueur de 245 si les états sont ajoutés à la frontière dans le même ordre que la liste retournée par la méthode* getSuccessors*, ou 146 si les états sont insérés dans l'ordre inverse.*

4. Recherche en largeur (breadth-first search)

Implémentez une recherche en largeur dans la fonction breadthFirstSearch du fichier search.py. Encore une fois, l'algorithme de recherche dans un graphe doit éviter d'explorer les états déjà visités. Le code peut être testé de la même façon que l'algorithme de recherche en profondeur :

```
~ $ python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

~ $ python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

Note *: Si l'algorithme a été implémenté de façon générique, il devrait pouvoir résoudre le jeu de taquin à 8 cases sans modification*

```
~ $ python eightpuzzle.py
```

5. Recherche à coût uniforme (uniform-cost search)

Même si l'algorithme de recherche en largeur réussi à trouver un chemin ayant un nombre minimal d'actions pour atteindre le but, il pourrait être intéressant de trouver de meilleurs chemins dans d'autres contextes. Considérez par exemple les labyrinthes mediumDottedMaze et mediumScaryMaze.

En changeant la fonction de coût, il est possible d'encourager Pac-Man à trouver des chemins différents. Par exemple, les déplacements dans les zones où il y a des fantômes pourraient avoir un coût plus grand, les zones où il y a beaucoup de gommes à manger pourraient avoir un coût moindre, et un agent rationnel devrait ajuster son comportement en conséquence.

Implémentez l'algorithme de recherche à coût uniforme dans la fonction uniformCostSearch du fichier search.py.

```
~ $ python2.7 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=uniformCostSearch
```

6. Recherche heuristique A*: 3 points

Implémentez l'algorithme de recherche heuristique dans un graphe A* dans la fonction **aStarSearch** du fichier **search.py.** L'algorithme A* prend une fonction heuristique en paramètre. Une fonction heuristique requiert deux paramètres : un état dans le problème de recherche et le problème lui-même. La fonction **nullHeuristic** dans search.py montre un exemple trivial.

Vous pouvez tester votre implémentation de A* en l'utilisant pour trouver un chemin vers une position donnée dans un labyrinthe en utilisant la distance de Manhattan comme fonction heuristique (déjà implémentée dans la fonction manhattanHeuristic du fichier searchAgents.py) :

```
~ $ python pacman.py -l bigMaze -z .5 -p SearchAgent -a
 fn=astar,heuristic=manhattanHeuristic
```

7. Soumission :

Remettez uniquement le fichier **search.py**.

8. Ressources :

Voici quelques ressources sur le langage Python qui pourraient vous aider dans votre travail

- Tutoriel Python par Marc-Alexandre Côté et Hugo Larochelle
- Documentation officielle de Python 2.7.13

9. Références :

- Projets Pac-Man du Cours « Introduction to Artificial Intelligence » de l'université Berkeley, Californie, USA, 2020.
  Cours : https://inst.eecs.berkeley.edu/~cs188/sp21/
  Projets : https://inst.eecs.berkeley.edu/~cs188/sp19/projects.html