Report LINFO1361: Assignment 1

Group N°G052

Student1: Bette Jonas

Student2: Huet Anatole

February 28, 2024

1 Python AIMA (3 pts)

1. In order to perform a search, what are the classes that you must define or extend? Explain precisely why and where they are used inside a *tree_search*. Be concise! (e.g. do not discuss unchanged classes). **(1 pt)**

Pour effectuer une recherche, il faut étendre la classe *Problem* avec les méthodes *actions*, *result* et *goal_test*. Elles sont utilisées dans *tree_search* pour connaître ce que PacMan peut faire, fera à chaque état et s'il a atteint son but.
Il faut définir la classe *Node* qui est utilisée dans *tree_search* pour se souvenir de l'état actuel, de l'état précédent et du chemin qui a été faite pour arriver à cet état.

2. Both *breadth_first_graph_search* and *depth_first_graph_search* are making a call to the same function. How is their fundamental difference implemented (be explicit)? **(0.5 pt)**

Breadth_first_graph_search utilise une Queue qui est une structure FIFO, tandis que *depth_first_graph_search* utilise une Stack qui elle est une structure LIFO. Ce ne sont donc pas les mêmes valeurs qui en resortiront, la première sort les valeurs dans l'ordre où elles sont entrées, tandis que le second sort les dernières valeurs entrées en premier.

3. What is the difference between the implementation of the *graph_search* and the *tree_search* methods and how does it impact the search methods? **(0.5 pt)**

Un *graph_search* va garder en mémoire les états déjà visités pour ne pas les revisiter, évitant ainsi une boucle infinie.
Un *tree_search* ne le fait pas et de ce fait requiert moins de mémoire.

4. What kind of structure is used to implement the *reached nodes minus the frontier list*? What properties must thus have the elements that you can put inside the reached nodes minus the frontier list? **(0.5 pt)**

La structure utilisée est un *set* ou un *dictionnaire*. Les éléments qui peuvent être mis dedans doivent être uniques et immuables. On peut y accéder en vérifiant si le hash de l'élément est présent dans le set ou dans la clé du le dictionnaire.

5. How technically can you use the implementation of the reached nodes minus the frontier list to deal with symmetrical states? (hint: if two symmetrical states are considered by the algorithm to be the same, they will not be visited twice) **(0.5 pt)**

Comme le *set* ou le *dictionnaire* ne peut contenir qu'une seule fois un élément, si on a deux états symétriques, ils auront le même hash/clé et ne seront donc pas ajoutés dans la structure.

2 The PacMan Problem (17 pts)

(a) **Describe** the set of possible actions your agent will consider at each state. Evaluate the branching factor**(1 pt)**

La taille d'une grille est de "n x m", le nombre d'actions possibles maximales est de "n + m - 2". En effet, sans mur, PacMan peut se déplacer en ligne droite jusqu'au bord de la grille. Donc, sans compter l'emplacement où il se situe, il a "n + m - 2" cases où il peut s'arrêter.
Le facteur de branchement quant à lui est aussi de "n + m - 2". On peut avoir maximum "n + m - 2" positions pour PacMan donc si on construit un arbre, chaque noeud aura au maximum "n + m - 2" enfants.

(b) How would you build the action to avoid the walls? **(1 pt)**

Pour éviter les murs, on peut simplement vérifier si la case où PacMan veut se déplacer est un mur. Si c'est le cas, on arrête la vérification des espaces libres et on retourne celles qu'on vérifiées.

2. **Problem analysis.**

   (a) Explain the advantages and weaknesses of the following search strategies **on this problem** (not in general): depth first, breadth first. Which approach would you choose? **(2 pts)**

L'avantage de *breadth_first* est qu'il parcourt toutes les cases donc il est assuré de trouver la solution optimale. Il peut cependant mettre beaucoup de temps à trouver la solution si la grille est grande. De plus, il utilise beaucoup de mémoire qui risque de ne pas permettre la fin de son exécution.
Depth_first quant à lui se dirige directement vers les cases les plus éloignées de la position initiale. Il sera plus rapide si les fruits sont éloignés de la position de PacMan. Il requiert aussi moins de mémoire car il parcourt en génréral moins de case.
Pour ce problème, nous choisirons d'utiliser *breadth_first* car on a un petit nombre d'états.

   (b) What are the advantages and disadvantages of using the tree and graph search **for this problem**. Which approach would you choose? **(2 pts)**

En utilisant un graphe, on évite de visiter plusieurs fois le même état, ce qui est un avantage car on gagne du temps. Cependant, on utilise plus de mémoire pour stocker les états déjà visités.
En utilisant un arbre, on risque de tomber dans une boucle infinie car on ne garde pas en mémoire les états déjà visités pour gagner de la place. On peut donc se retrouver à visiter plusieurs fois le même état .
Pour ce problème, on choisirait d'utiliser un graphe car on a un nombre fini d'états et on ne risque pas de tomber dans une boucle infinie.

3. **Implement** a PacMan solver in Python 3. You shall extend the *Problem* class and implement the necessary methods -and other class(es) if necessary- allowing you to test the following four different approaches:

   - *depth-first tree-search (DFSt)*;
   - *breadth-first tree-search (BFSt)*;
   - *depth-first graph-search (DFSg)*;
   - *breadth-first graph-search (BFSg)*.

   Experiments must be realized (*not yet on INGInious!* use your own computer or one from the computer rooms) with the provided 10 instances. Report in a table the results on the 10 instances for depth-first and breadth-first strategies on both tree and graph search (4 settings above). Run each experiment for a maximum of 1 minute. You must report the time, the number of explored nodes as well as the number of remaining nodes in the queue to get a solution. **(4 pts)**

Inst.	BFS						DFS					
	Tree			Graph			Tree			Graph		
	T(s)	EN	RNQ	T(s)	EN	RNQ	T(s)	EN	RNQ	T(s)	EN	RNQ
i_01	0.07	116	1421	0.005	9	106	Ø	Ø	Ø	Ø	Ø	Ø
i_02	0.03	96	861	0.003	9	86	Ø	Ø	Ø	Ø	Ø	Ø
i_03	357.43	1354575	10124430	3804	159276	1195298	Ø	Ø	Ø	Ø	Ø	Ø
i_04	156.61	242209	2774391	1221	19247	222961	Ø	Ø	Ø	Ø	Ø	Ø
i_05	3.29	6849	66352	0.3	692	6156	Ø	Ø	Ø	Ø	Ø	Ø
i_06	0.05	283	2282	0.007	32	250	Ø	Ø	Ø	Ø	Ø	Ø
i_07	0.47	2157	16410	0.06	247	1909	Ø	Ø	Ø	Ø	Ø	Ø
i_08	0.008	89	451	0.001	16	72	Ø	Ø	Ø	Ø	Ø	Ø
i_09	0.02	76	581	0.003	11	64	Ø	Ø	Ø	Ø	Ø	Ø
i_10	0.03	96	861	0.003	9	86	Ø	Ø	Ø	Ø	Ø	Ø

T: Time — **EN**: Explored nodes — **RNQ**: Remaining nodes in the queue — **Ø**: Timeout

4. **Submit** your program (encoded in **utf-8**) on INGInious. According to your experimentations, it must use the algorithm that leads to the best results. Your program must take as inputs the four numbers previously described separated by space character, and print to the standard output a solution to the problem satisfying the format described in Figure 3. Under INGInious (only 1 minute timeout per instance!), we expect you to solve at least 12 out of the 15 ones. **(6 pts)**

5. **Conclusion.**

   (a) How would you handle the case of some fruit that is poisonous and makes you lose? **(0.5 pt)**

Si on voit que le fruit est empoisonné, on peut simplement le considérer comme une case vide et donc passer au dessus sans s'y arrêter.
Si on peut s'arrêter sur la case où se trouve le fruit sans obligatoirement le manger alors on peut se servir de cette case pour avoir un chemin plus court dans les situations où la case de ce fruit empoisonné est un point où le PacMan aurait pû s'arrêter au lieu de contourner le fruit.

   (e) Do you see any improvement directions for the best algorithm you chose? (Note that since we're still in uninformed search, *we're not talking about informed heuristics*). **(0.5 pt)**

Avec un *iterative_deepening_depth_first_search*, on pourrait trouver une solution nécessitant moins de mémoire que *breadth_first_graph_search* (l'algorithme que nous avons choisi) car c'est un algorithme qui a une complexité spacialle réduite.