Wumpus IA - Phase 2

Ulysee Brehon, Luis Enrique González Hilario

June 2020

Sommaire

1 Introduction

La deuxième phase du projet consiste essentiellement à planifier le déplacement de l'agent sur la carte. Dans cette partie du projet, les algorithmes vus dans le cours IA02 seront utilisés.

2 Environnement

Notre projet (phase 1 et phase 2) est composé de deux fichiers: **cartographie.py** et **planification.py**.
La variable **gophersat_exec** contient le chemin gophersat pour linux. En cas d'exécution sur Windows, remplacez-le simplement par: **./lib/gophersat**

3 DFS et BFS

1. **estLibre (i:int, j:int) – bool** Fonction booléenne qui dit True si les elements dans la carte ne correpondent pas à (W: Wumpus, P: Pit) et false sinon.

2. **parcours_largeur (size:int) – List** On commence par explorer un nœud source, puis ses successeurs, puis les successeurs non explorés des successeurs, etc

3. **parcours_profondeur (size:int) – List** Son fonctionnement consiste à étendre chacun et chacun des nœuds qu'il localise, de façon récurrente, sur un chemin spécifique. Lorsqu'il n'y a plus de nœuds à visiter sur ce chemin, il retourne (Backtracking), donc il répète le même processus avec chacun des frères du nœud déjà traité.

4. **chemin_largeur (size:int, init:Tuple[int, int], but:Tuple[int, int]) – Tuple et chemin_profondeur (size:int, init:Tuple[int, int], but:Tuple[int, int]) – List** Chemin allant de la case initiale à la case but.

4 Heuristique: Distance de Manhattan

1. **distance_manhattan (depart:Tuple[int, int], but:Tuple[int, int]) – int** La distance de Manhattan est obtenu par la formule suivant: abs(x0x1)+abs(y0y1)

2. **get_minimum_distance_state (successeurs:Tuple, but:Tuple)** On calcule la distance de Manhattan entre les candidats et le but. Enfin, nous sélectionnons la distance minimale et la présentons sous la forme ((i, j), distance)

5 Algorithme Glouton

1. **glouton (size:int, init:Tuple[int, int], but:Tuple[int, int]) – List**

   Dans cette version, on recommence a calculer un chemin si jamais nous tombons sur un cul-de-sac. C'est très gourmand et ça ne garantis pas le meilleur chemin mais si il existe un chemin, il le trouvera.

2. **successeurs_glouton (position:Tuple[int, int], size:int, dead**$_e nd$: $Dict, chemin : Tuple) - -Tuple$

   Cette fonction retourne les successeurs qui ne sont pas des culs-de-sac.

3. **get_gold_position (size:int)** Cette fonction vise à obtenir une liste des positions d'or.

6 Algorithme Alpha Star

1. **a_etoile (size:int, init:Tuple, but:Tuple)** A chaque iteration, on calcule le cout pour aller de l'etat initial vers la l'etat cible à partir de l'etat courant. Si jamais un chemin avait deja été trouvé vers la cible, on prend

le chemin le moins couteux. Pour cela, On utilise une distance de Manhattan ainsi qu'un cout cumulé depuis l'etat initial tel que : g = f + h avec g le cout total, f le cout depuis l'etat inital et h notre heuristique.