Partie 3

CARTAYA Agustin

Algorithme du chasseur

Algorithme du chasseur 0:
Dans cet algorithme le chasseur se déplace au hasard sans chercher à se rapprocher de sa proie, il est implémenté pour se faire une idée du pire chasseur, il est représenté par le chiffre "0" car il ne remplit pas les conditions proposées dans l'énoncé

Fonctionnement
1) les CVC (casses voisines connexes) du chasseur sont calcules
2) seuls les CVC valides (ceux qui sont différentes a -1) sont pris et l'un d'entre eux est choisi au hasard pour déplacer le chasseur

Algorithme du chasseur 1:
Dans cet algorithme le chasseur cherche à chaque étape à minimiser la distance entre lui-même et sa proie, s'il y a plusieurs chemin qui minimisent la distance, l'un d'entre eux est choisi au hasard

Fonctionnement:
I) la distance entre le chasseur et la proie est calculée avec la méthode "distMin"
2) les CVC du chasseur sont calculés
3) Seuls les CVC valides et qui réduisent la distance entre le chasseur et la proie sont pris (on fait appel a la méthode "distMin" pour calculer les distances entres les CVC du chasseur et la proie)
4) Une de ces cases est choisie au hasard et le chasseur est déplacé vers elle

Algorithme du chasseur 2:
Cet algorithme recherche le chemin le plus court entre le chasseur et la proie, et le chasseur suit toujours ce chemin.

pour les proies peu intelligentes, il est très efficace car il minimise le temps d'exécution et les résultats sont presque les mêmes que l'algorithme précédent, mais pour les proies légèrement intelligentes, cet algorithme n’est pas très efficace.

Ne pas utiliser cette algorithme pour les labyrinthes de plus de 30 * 35 casses car il y a un débordement de mémoire

Fonctionnement
1) vérifier que le labyrinthe n'a pas plus de 30 * 35 casses pour pouvoir l'exécuter
2) le chemin le plus court entre le chasseur et la proie est calculé
3) Le chasseur est déplacé en suivant ce chemin

Algorithme de la proie

Algorithme de la proie 1:
C'est un algorithme de type proie aléatoire, puisque la proie se déplace au hasard sans chercher à échapper du chasseur

Fonctionnement
1) les CVC de la proie sont calcules
2) seuls les CVC valides sont pris et l'un d'entre eux est choisi au hasard pour déplacer la proie

Algorithme de la proie 2:
Dans cet algorithme la proie cherche à chaque pas à augmenter la distance entre elle et son chasseur, s'il y a plusieurs cumin qui augmentent la distance, l'un d'entre eux est choisi au hasard

Fonctionnement:
1) les CVC de la proie sont calcules
2) la distance entre le chasseur et la proie est calculée avec la méthode "distMin"
3) Seuls les CVC valides et qui augmentent ou mantient la distance entre le chasseur et la proie sont pris (on fait appel a la méthode "distMin" pour calculer les distances entres les CVC du chasseur et la proie)
4) Un de ces casses est choisi au hasard pour déplacer la proie

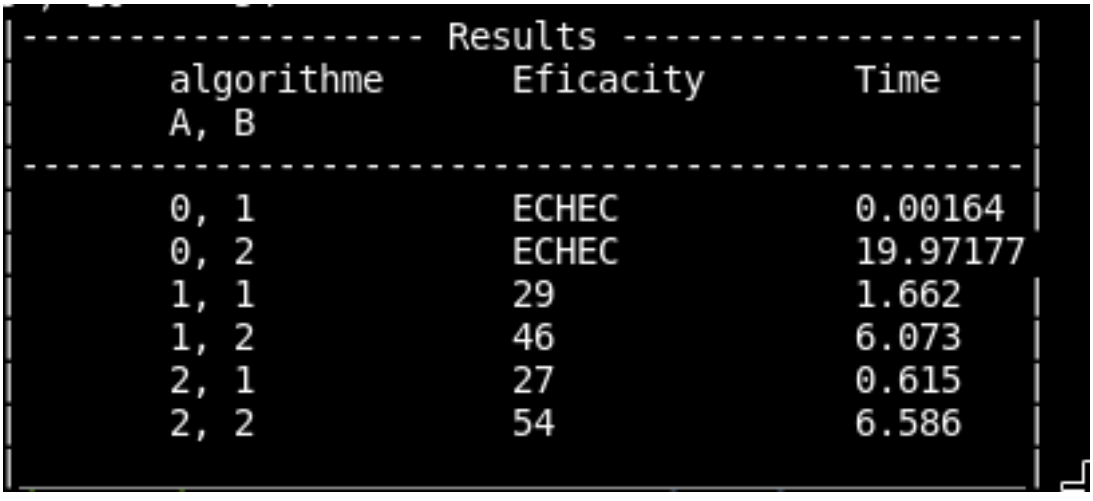
Obtention des CVC
pour obtenir les CVC d'une casse d'identifiant "id", la méthode "voisinsConnexesID" est utilisée, elle accepte comme paramètre l'id de la cellule dans laquelle les voisins connexes doivent être recherchés et renvoie un tableau de 4 éléments avec les CVC de cette casse

Fonctionnement:
1) un tableau à 4 positions est créé dans le tas et rempli avec des -1
2) Les cases au-dessus, à gauche, à droite et en dessous de la case "id" sont vérifiés (en suivant cet ordre) et si celles-ci existent et ne sont pas des murs, elles sont ajoutées au tableau
3) l'adresse du tableau est retourne par la fonction

Obtention du chemin le plus court
Pour obtenir le chemin le plus court entre un point A et un point B du labyrinthe, on utilise la méthode "cheminMin" qui demande en paramètre l'identifiant du point A et du point B et renvoie en résultat une liste contenant comme premier élément le nombre de casses entre le point A et le point B et les autres éléments sont l'identifiant des cellules qui composent ce chemin (A et B sont inclus)

Fonctionnement:
1) convertir le labyrinthe en un graphe de contiguïté à l'aide de la fonction "lab2Graphe" (procédure expliquée dans le code)
2) la distance minimale entre A et B est calculée à partir de la liste de contiguïté en utilisant la méthode "dijkstra" (procédure expliquée dans le code)

Tests d'algorithmes
Pour observer l'efficacité et la vitesse de tous les algorithmes utilisés, une fonction "testAlgos" a été créée qui teste chacun des algorithmes en utilisant la fonction "evalue" et à la fin montre un tableau avec les résultats de chaque algorithme



Results

algorithme A, B	Eficacity	Time
0, 1	ECHEC	0.00164
0, 2	ECHEC	19.97177
1, 1	29	1.662
1, 2	46	6.073
2, 1	27	0.615
2, 2	54	6.586