Rapport projet Graphe

Bordier

De Vaugelade

Zhao

Spécification fonctionnelle

Hors_Limite : case ➔ Booléen

Vérifie que la case placée en paramètre est dans la map ou en dehors

Renvoie True si la case est en dehors et False sinon

Mur_Eau: case ➔ Booléen

Vérifie que la case placée en paramètre est un mur ou de l'eau ou pas.

Renvoie True si la case est un mur ou de l'eau et False sinon

Pas_Obstacle : case ➔Booléen

Vérifie que la case placée en paramètre est un obstacle en faisant appel aux deux fonctions précédentes

Renvoie True si la case est un et False sinon

Distance_Heuristique: case X case ➔ int

Calcule la distance en nombre de case entre la case de départ et la case d'arrivée

Renvoie la distance entre la case et la fin

Voisins_Proches: case ➔ liste de case

Vérifie si les voisins de la case placée en paramètre sont oui ou non des cases accessibles.

Renvoie la liste des cases voisines qui sont accessibles (pas des obstacles)

Distance_Voisin: case X case ➔ int

Permet de calculer le coût pour passer d'une case à son voisin.

On connaît le coût pour parcourir une case donc on fait la moyenne des coûts des deux cases.

Renvoie la moyenne des coûts de parcours d'une case et de son voisin.

Reconstruire_Chemin: liste de cases X case ➔ liste de case

Reconstruit le chemin du point d'arrivée au point de départ

Renvoie la liste des cases pour aller du point de départ au point d'arrivée dans l'ordre inverse.

Astar : case X case ➔ liste de case

Si au moins une route existe entre la case de départ et la case d'arrivée alors on renvoie le chemin qui coûte le moins cher (dans le bon ordre). Sinon on renvoie qu'il n'y a pas de chemin.

Représentation des données

Nous avons décidé de représenter la map par un tableau à deux dimensions et chaque case est représentée par le coût pour la traverser : 5 pour les chemins et les ponts, 20 pour l'herbe, 70 pour la forêt et -1 pour les murs et l'eau.

Pour notre représentation les points de départ sont

(2,0),(4,7),(9,31),(22,31),(29,2),(30,18),(31,27),(41,7),(40,10),(42,18),(40,20),(42,22),(35,34)

Et le point d'arrivée est (21,5)

Le nombre de colonnes est de 43 et le nombre de lignes est de 37

Ce qui fait que la représentation de la map prend 3 pages.

Voici la première ligne comme exemple :

[20, 20, 20, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 20, 20, 20, 20, 20, 20, 20, 20, 20],

Vous trouverez la map dans le fichier astar.py

Définitions des fonctions

Nous avons choisi de coder les fonctions en python.

```
def Hors_Limite(case):
    x,y = case
    return x<0 or x>=TAILLE_COL or y<0 or y>=TAILLE_LIGNE  # On vérifie qu'on est bien dans la map


def Mur_Eau(case):
    x,y = case
    return map[y][x]==-1 # on vérifie que la case n'est pas inaccessible


def Pas_Obstacle(case):
    return not (Hors_Limite(case) or Mur_Eau(case))


def Distance_Heuristique(case_debut,case_fin):
    dist=abs(case_fin[0]-case_debut[0])+abs(case_debut[1]-case_fin[1])
    return dist                                  # On renvoie la distance en allant tout droit (en x et en y) vers la fin


def Voisins_Proches(case):
    x, y = case
    voisins = [(x-1,y),(x+1,y),(x,y-1),(x,y+1)]  # On repère les 4 voisins de la case actuelle
    return filter(Pas_Obstacle,voisins)          # On ne garde que les voisins que l'on peut parcourir


def Distance_Voisin(case,voisin):
    return (map[case[1]][case[0]]+map[voisin[1]][voisin[0]])/2  # On calcule le coût pour rejoindre le voisin


 def Reconstruire_Chemin(CasesPrecedentes,case_Actuelle):
    Chemin = [case_Actuelle]
    while case_Actuelle in CasesPrecedentes:
        case_Actuelle = CasesPrecedentes[case_Actuelle]
        Chemin.append(case_Actuelle)
    return Chemin                                # On a récupéré toutes les cases qui ont permis d'aller du départ jusqu'à l'arrivée
```

```
def Astar(depart,arrivee):
    CasesEvaluees = []                          # la liste des cases évaluées.
    CasesEncoursEval = [depart]                 # liste des cases qu'on est en train d'évaluer
    CasesPrecedentes = {}                       # la dernière case qu'on a visité
    d_score = {}                                # coût estimé depuis le début.
    d_score[depart] = 0
    f_score = {}                                #coût estimé pour arriver à la fin

    f_score[depart] = Distance_Heuristique(depart, arrivee)

    while CasesEncoursEval:
        case_Actuelle = min((f_score[noeud], noeud) for noeud in CasesEncoursEval)[1]
        if case_Actuelle == arrivee:
            return Reconstruire_Chemin(CasesPrecedentes, arrivee)          # On est arrivé
      CasesEncoursEval.remove(case_Actuelle)
      CasesEvaluees.append(case_Actuelle)
      for voisin in Voisins_Proches(case_Actuelle):
        if voisin in CasesEvaluees:
            continue                            #  ignore les cases déjà évaluées

        tentative_d_score = d_score[case_Actuelle] + Distance_Voisin(case_Actuelle,voisin)
                                                # calcule de la longueur du chemin.
        if voisin not in CasesEncoursEval:      # découverte d'une nouvelle case
            CasesEncoursEval.append(voisin)     # on l'ajoute à la liste des cases à évaluer
        elif tentative_d_score >= d_score[voisin]:
            continue

        CasesPrecedentes[voisin] = case_Actuelle
        d_score[voisin] = tentative_d_score
        f_score[voisin] = d_score[voisin] + Distance_Heuristique(voisin, arrivee)
return "pas de chemin"                          # on n'est pas arrivé
```

Représentation des résultats

Voici des exemples de représentations des résultats de notre algorithme.

A chaque fois le premier couple est le point de départ

----------Chemin----------

[(2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9), (1, 9), (1, 10), (1, 11), (2, 11), (3, 11), (4, 11), (4, 12), (4, 13), (5, 13), (6, 13), (7, 13), (8, 13), (9, 13), (10, 13), (10, 14), (10, 15), (10, 16), (10, 17), (10, 18), (11, 18), (12, 18), (12, 19), (13, 19), (14, 19), (15, 19), (16, 19), (17, 19), (18, 19), (19, 19), (20, 19), (20, 18), (21, 18), (21, 17), (21, 16), (21, 15), (21, 14), (21, 13), (21, 12), (21, 11), (21, 10), (21, 9), (21, 8), (21, 7), (21, 6), (21, 5)]

----------Chemin----------

[(4, 7), (4, 8), (4, 9), (5, 9), (6, 9), (6, 10), (6, 11), (7, 11), (8, 11), (9, 11), (10, 11), (10, 12), (10, 13), (10, 14), (10, 15), (10, 16), (10, 17), (10, 18), (11, 18), (12, 18), (12, 19), (13, 19), (14, 19), (15, 19), (16, 19), (17, 19), (18, 19), (19, 19), (20, 19), (20, 18), (21, 18), (21, 17), (21, 16), (21, 15), (21, 14), (21, 13), (21, 12), (21, 11), (21, 10), (21, 9), (21, 8), (21, 7), (21, 6), (21, 5)]

----------Chemin----------

[(9, 31), (9, 30), (9, 29), (9, 28), (9, 27), (9, 26), (9, 25), (9, 24), (10, 24), (11, 24), (11, 23), (12, 23), (13, 23), (14, 23), (15, 23), (16, 23), (16, 22), (16, 21), (16, 20), (16, 19), (17, 19), (18, 19), (19, 19), (20, 19), (20, 18), (21, 18), (21, 17), (21, 16), (21, 15), (21, 14), (21, 13), (21, 12), (21, 11), (21, 10), (21, 9), (21, 8), (21, 7), (21, 6), (21, 5)]

----------Chemin----------

[(22, 31), (22, 30), (22, 29), (21, 29), (21, 28), (21, 27), (20, 27), (20, 26), (20, 25), (20, 24), (20, 23), (20, 22), (20, 21), (20, 20), (20, 19), (20, 18), (21, 18), (21, 17), (21, 16), (21, 15), (21, 14), (21, 13), (21, 12), (21, 11), (21, 10), (21, 9), (21, 8), (21, 7), (21, 6), (21, 5)]

----------Chemin----------

[(29, 2), (28, 2), (27, 2), (26, 2), (26, 3), (26, 4), (26, 5), (26, 6), (25, 6), (24, 6), (23, 6), (22, 6), (21, 6), (21, 5)]

----------Chemin----------

[(30, 18), (29, 18), (28, 18), (27, 18), (27, 19), (26, 19), (25, 19), (25, 18), (24, 18), (23, 18), (22, 18), (21, 18), (21, 17), (21, 16), (21, 15), (21, 14), (21, 13), (21, 12), (21, 11), (21, 10), (21, 9), (21, 8), (21, 7), (21, 6), (21, 5)]

Le reste peut être vu grâce au fichier astar.py