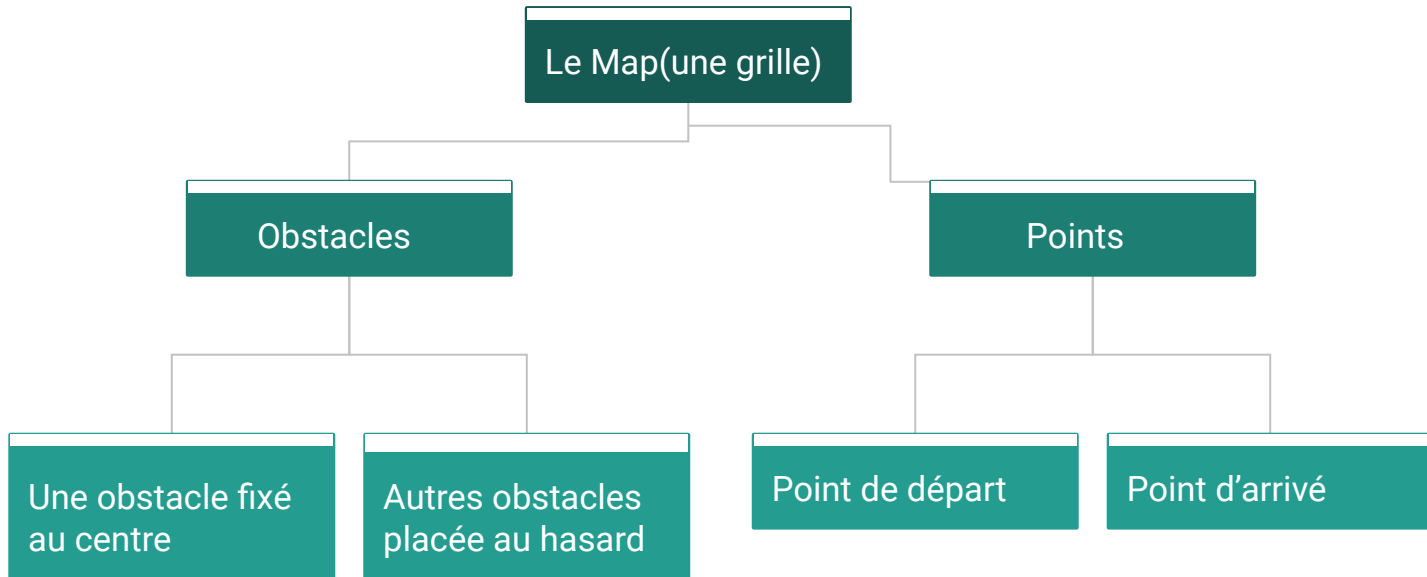


Le plan de la présentation

- Introduction du sujet choisi
- Explication de l'algorithme A* & La recherche de la fonction heuristique
- Explication sur OpenList
- Autres Parties des codes
- Tests & Applications
- Conclusion



Introduction du sujet choisi





Algorithme A*

$$f(n) = g(n) + h(n)$$

L'estimation du coût
de la distance entre
le point départ et le
point arrivé

Le coût de la
distance entre
le point départ
et le point n

Le coût de la
distance entre le
point n et le point
arrivé



La fonction de la distance heuristique

Distance de Manhattan

4 directions (vers le haut, vers le bas, vers la gauche, vers la droite)
Les déplacements sont horizontaux et verticaux.

Distance diagonale

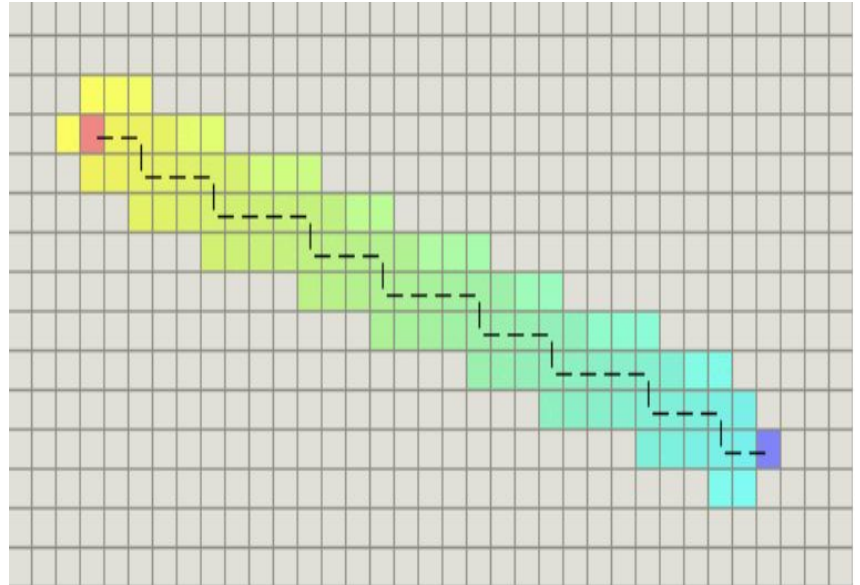
8 directions
Les déplacements sont en diagonale.

Distance Euclidienne

Ce qui nous permet de se déplacer à un angle quelconque
(au lieu de directions de la grille)

Distance de Manhattan

Heuristique de fonction (noeud) =
dx = abs(noeud.x - goal.x)
dy = abs(noeud.y - goal.y)
retourner D*(dx+dy)

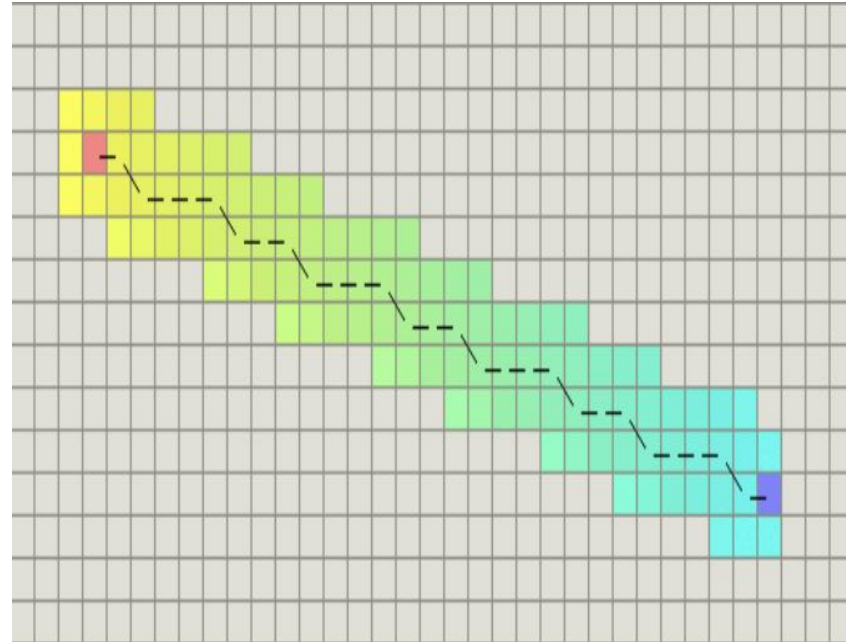


D : comme le coût minimal de partant d'un point choisi à un point juste à côté de cet point.

Distance diagonale

Heuristique de fonction (noeud) =
dx = abs(noeud.x - goal.x)
dy = abs(noeud.y - goal.y)
retourner $D^* (dx + dy) + (D2 - 2 * D) * \min (dx , dy)$

$D2 = \text{sqrt}(D)$

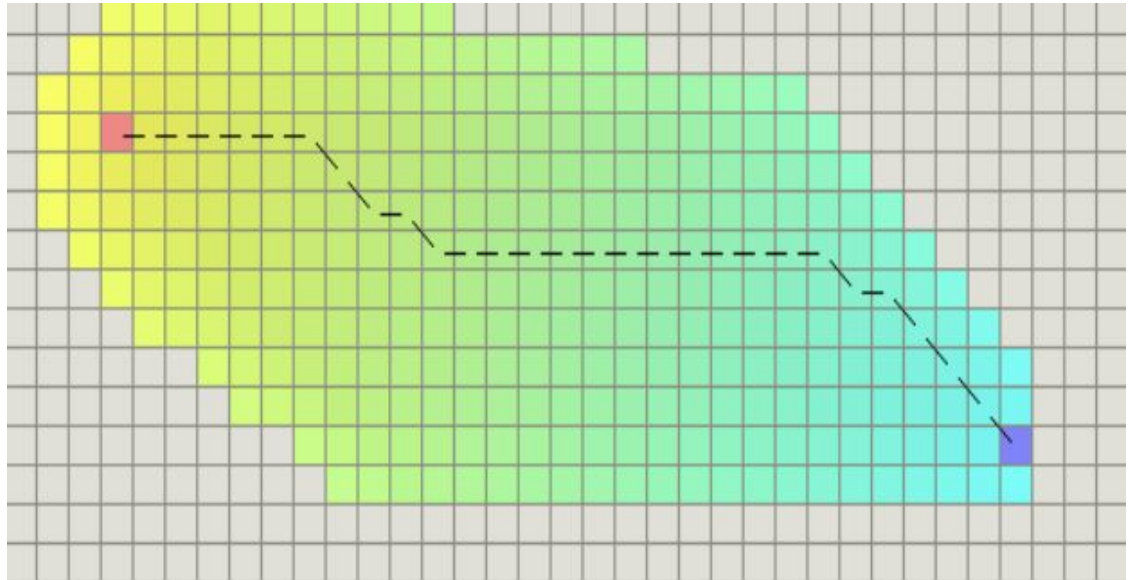


Code

```
16 class AStar:
17
18     """ 构造函数 """
19
20     def __init__(self, Map_size, obstacle_point, p_start, p_end):
21         self.Map_size = Map_size
22         self.obstacle_point = obstacle_point
23         self.p_start = p_start
24         self.p_end = p_end
25         self.open_set = []
26         self.close_set = []
27
28
29     """ 节点到起点的移动代价, 对应g(n) """
30
31
32     def GCost(self, p):
33         h_diagonal = min(np.abs(self.p_start.x - p.x - 1), np.abs(self.p_start.y - p.y - 1))
34         h_stright = np.abs(self.p_start.x - p.x - 1) + np.abs(self.p_start.y - p.y - 1)
35         # Distance to start point
36         return h_stright + (np.sqrt(2) - 2) * h_diagonal
37
38 # 节点到终点的启发函数, 对应h(n)
39
40     def HCost(self, p):
41         h_diagonal = min(np.abs(self.p_end.x - p.x - 1), np.abs(self.p_end.y - p.y - 1))
42         h_stright = np.abs(self.p_end.x - p.x - 1) + np.abs(self.p_end.y - p.y - 1)
43         # Distance to start point
44         return h_stright + (np.sqrt(2) - 2) * h_diagonal
45
46 # 代价总和, 即对应f(n)
47
48     def TotalCost(self, p):
49         return self.GCost(p) + self.HCost(p)
```


Distance euclidienne

Sur une grille carrée qui permet de se déplacer à un angle quelconque (au lieu de directions de la grille)





Explication sur OpenList

1. Quelles points ne sont pas au dedans ?
2. Conditions d'utilisation :
 - a. Si Open est vide
 - b. Si un sous noeud m de n (où on est) est pas au dedans
 - c. Si on a trouvé le point m tel que $f(m)$ atteint son minimum
 - d. Si on arrive au point final
3. Intérêt de OpenListe ?
4. Comment peut on appeler OpenList dans le code?



My_Map

- ❑ Rappeler la dimension de la carte : size
- ❑ Créer des obstacles et enregistrer leurs coordonnées dans une liste
 - ❑ Au milieu
 - ❑ Aléatoire
 - ❑ Aléatoire dans deux directions



Main

- ❑ Désigner la carte aléatoire
- ❑ Désigner les obstacles
- ❑ Mettre des couleurs pour chaque condition :
 - ❑ Gris pour les obstacles
 - ❑ **Bleu** pour le point départ et point accédés
 - ❑ **Rouge** pour le point d'arrivé
 - ❑ **Vert** pour chemin finale

Tests appliquées (1) - Erreurs d'entrées

```
##### Demander des informations a partir d'utilisateur

print("Donner la taille de map,le nb d'obstacle, separees par un espace:")
size, nbObstacle = map(int,input().split())


print("Donner ensuite le x et y point de depart, separees par un espace:")
print("x,y sont dans [0,%d]" % (size-1))
x_start, y_start = map(int,input().split())

if x_start < 0 or x_start > size-1 or y_start < 0 or y_start > size-1:
    raise AssertionError
else:
    p_start = point.Point(x_start, y_start)


print("Donner ensuite le x et y point d'arrive, separees par un espace:")
print("x,y sont dans [0,%d]" % (size-1))
x_end, y_end = map(int,input().split())
if x_end < 0 or x_end > size-1 or y_end < 0 or y_end > size-1:
    raise AssertionError
else:
    p_end = point.Point(x_end, y_end)
```

Nous précisons les contraintes suivantes:

le point de départ et le point d'arrivée ne doivent pas dépasser l'intervalle [0, Map.size-1].



```
Donner la taille de map,le nb d'obstacle, separees par un espace:
30 2
Donner ensuite le x et y point de depart, separees par un espace:
x,y sont dans [0,29]
-1 5
```



```
File "C:/Users/lssya/Desktop/A星/main.py", line 28, in <module>
    raise AssertionError
AssertionError
```

Tests appliquées (2) - Création une carte

```
plt.figure(figsize=(5, 5))
Map = My_Map.MyMap(size, nbObstacle) # create a map
obs_point = Map.getObstacles() # create the obstacles

# Draw the map with the white square and gray border
rec = Rectangle((i, j), width=1, height=1, edgecolor='gray', facecolor='w')
ax.add_patch(rec)

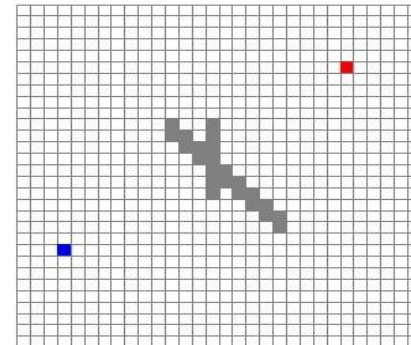
# Draw the obstacles with the gray square
rec = Rectangle((i, j), width=1, height=1, color='gray')
ax.add_patch(rec)

# draw the start point with the blue square
rec = Rectangle((x_start, y_start), width = 1, height = 1, facecolor='b')
ax.add_patch(rec)

# draw the start point with the red square
rec = Rectangle((x_end-1, x_end-1), width = 1, height = 1, facecolor='r')
ax.add_patch(rec)
```

Nous créons une carte en donnant les paramètres nécessaires suivantes:

Une carte de taille 30 avec 2 obstacles au hasards. Nous fixons aussi le point de départ (3, 8) et point d'arrivée (25, 28).



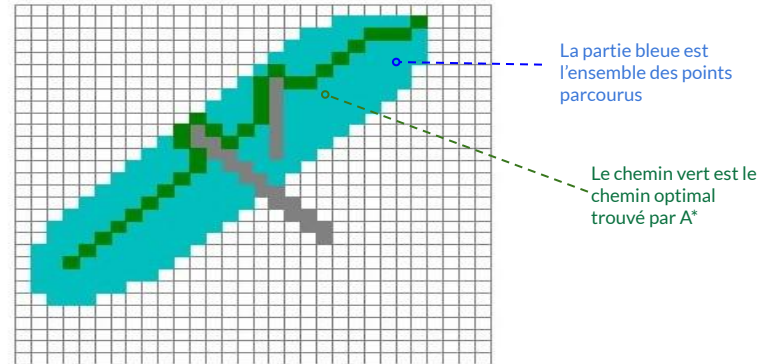
Tests appliquées (3) - Recherche un chemin optimal

```
#!/usr/bin/env python3
# %% Lancer l'algorithme de A*

a_star = A_star.AStar([size,size], obs_point, p_start, p_end)
a_star.RunAndSaveImage(ax, plt)
```

Nous testons si notre programme peut trouver un chemin optimal ou pas en utilisant la methode `RunAndSaveImage()` de la classe `A_star`.

De plus, nous allons voir le temps qu'il prend pour finaliser l'exécution.

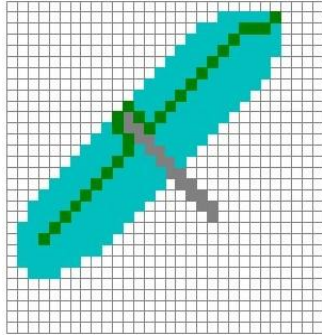


Tests appliquées (4) - Temps utilisé

Nous voulons comparer le temps utilisé si nous sommes dans une carte de même taille mais avec **différent nombre d'obstacles**.

Nous gardons aussi le même point de départ et le même point d'arrivée.

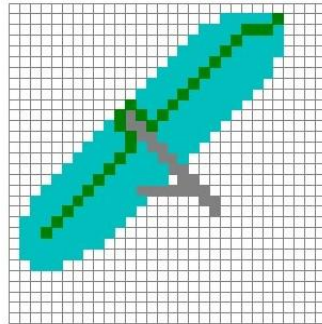
==== Algorithm finish in 8.7780 seconds



(1)- 1 obstacle
Temps utilises 8.7780s



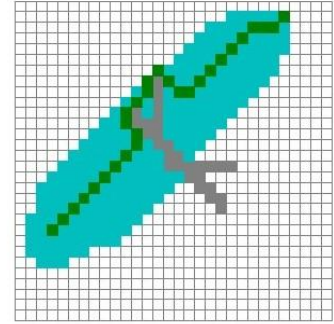
==== Algorithm finish in 8.8223 seconds



(2)- 2 obstacles
Temps utilises 8.8223s



==== Algorithm finish in 9.1845 seconds



(3)- 3 obstacles
Temps utilises 9.1845s

Applications (1) - Carte entre Technopôle et l'insa

Nous nous posons dans la situation suivante:

Je suis un nouvel étudiant de première année de L'INSA Rouen. Maintenant je suis arrivé à la station de Technopôle. Comment puis je arriver à l'INSA?



Une carte cherchée sur Google Map.

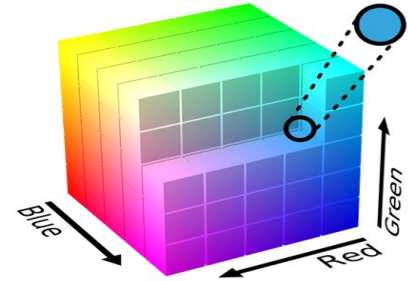
Applications (1) - Binarization d'une image



Nous voyons dans cette carte, le chemin accessible est coloré par blanc et les autres couleurs sont considérés comme des obstacles.

Dans la machine, une image est définie comme une superposition de 3 matrices de taille longueur*largeur*hauteur. Et le hauteur est composé par rouge, bleu et vert.

Ainsi, pour définir des obstacles, nous devons fixer deux points dans le RGB espace.



RGB space
Ref: https://en.wikipedia.org/wiki/Color_spaces_with_RGB_primaries

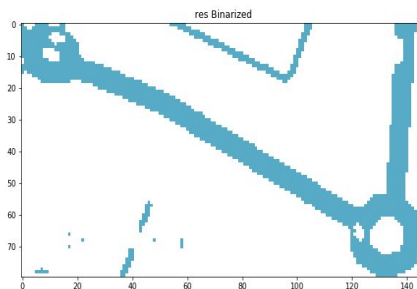
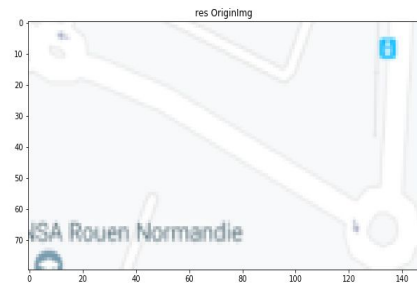
Applications (1) - Binarization d'une image

```
#!/usr/bin/env python
# Binarization
# black 0,0,0 white 255,255,255 gray x,x,x

th = 250
img_bin = (img[:, :, 0] > th) * (img[:, :, 1] > th) * (img[:, :, 2] > th)
fig = plt.figure(figsize=[10, 10])
img_bin = 1 - img_bin #反轉0 1
plt.imshow(img_bin, cmap='ocean', vmin=-3.5, vmax=1)
#plt.colorbar()
plt.title(file+' Binarized')
```

Dans notre exemple, nous obtenons les chemins possibles en posant une paramètre thresholds.

Nous fournirons aussi des intervalles utiles :
black (0,0,0) white(255,255,255) gray(x,x,x)

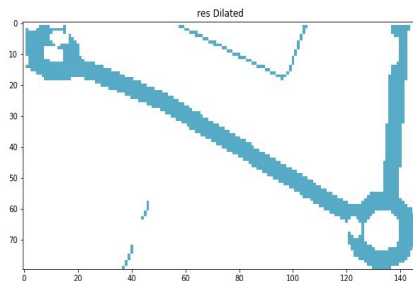
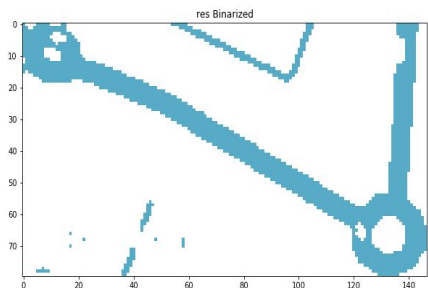


Applications (1) - Dilatation d'un chemin

Pour éviter des impasses possibles, nous allons fixer une distance de 2 pixels et la dilatation retourne la valeur maximum parmi des points adjacents. (0: not obstacle, 1: obstacle)

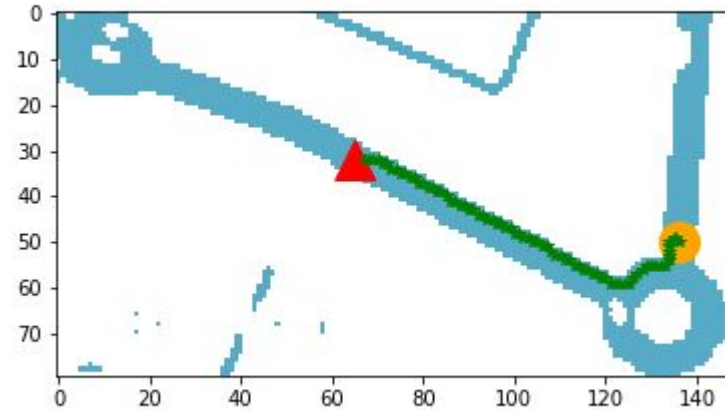
```
#!/usr/bin/env python3
# %% Dilation operator

w, h = img_bin.shape
half_ker = 1 # the half side length of conv kernel
bidiimg = np.ones((w,h))
for i in range(half_ker, w - half_ker + 1):
    for j in range(half_ker, h - half_ker + 1):
        bidiimg[i,j] = np.max(img_bin[i-half_ker:i+half_ker, j-half_ker:j+half_ker])
mymap = img_bin+bidiimg
```



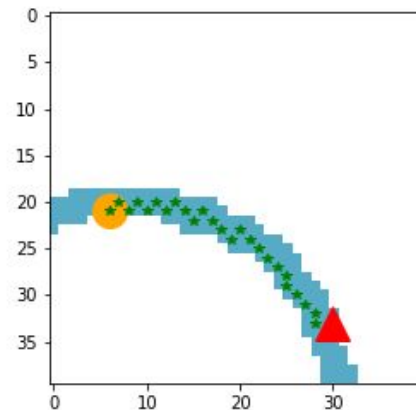
Applications (1) - Application A*

En utilisant le A* que nous avons écrit, nous avons réussi à trouver un chemin vert pour cet étudiant pour aller à l'INSA.



Applications (2) - Le Rond Point

De même procédure précédente, nous avons réussi de trouver un chemin optimal sur le Rond Point.





Conclusion

- ❑ Accomplissement
- ❑ Distribution du travaux
- ❑ Questions ?