

phase 2 Gemeni Pro

1er prompts utiliser:

“Vous avez dit

salut fait moi un programme en python qui permettrait de lire un fichier text que je vais donner avec des plusieurs point pour mettre dans un diagramme et ton but se sera de fait un diagramme de voronoi avec les point qui sont de le fichier mais il ya plusieurs contrainte a prendre en compte dans un premier temps tu na pas le droit au bibliothèque qui impory directemnt le diagramme de voronoi ni des outils qui sont justement fait pour ce digramme tu a doit a math,matplotlib ou des biblotehque d'affcihage comme draw svg. Dans un second temps tu devrat respecter plusieurs point comme par exemple L'application fournie par l'IA devra respecter le plus haut niveau de

bonnes pratiques de programmation. Elle inclura donc aussi notamment une

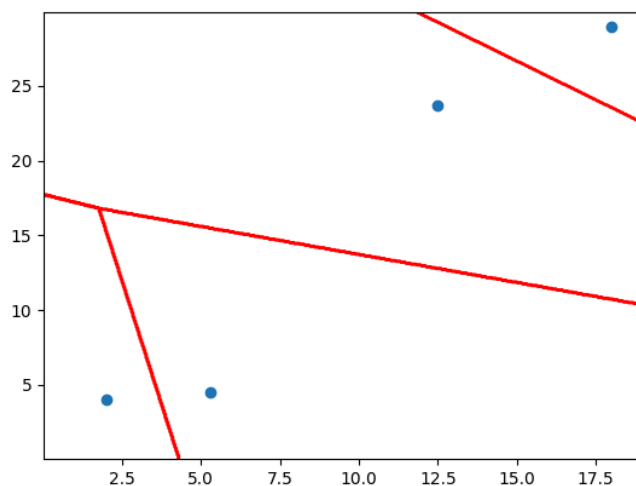
série de tests. Voici un exemple de diagramme attendu avec les point suivant 2,4

5.3,4.5

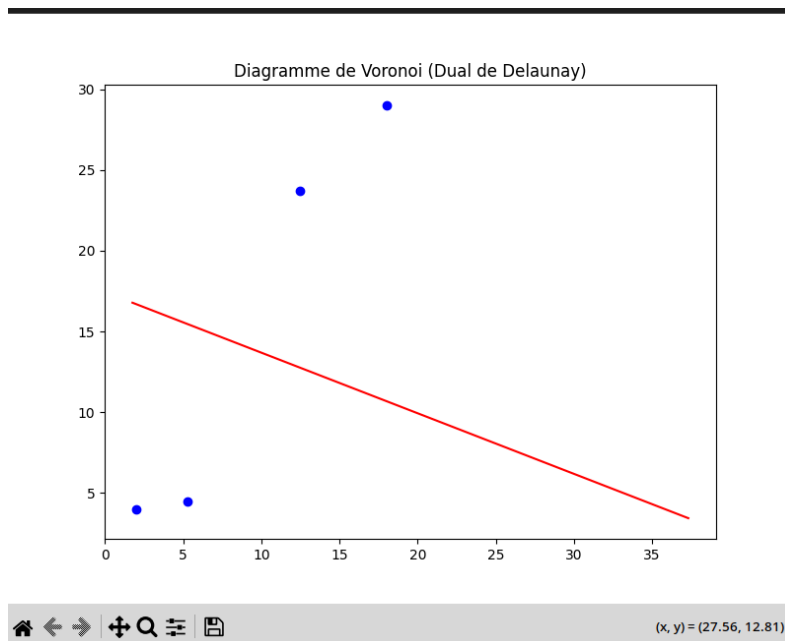
18,29

12.5,23.7 pour realsier cela comme tu na pas le droit au bibliotehque predéfini tu pourrai par exemple implementer une triangulation de delaunay ou implementer un algo de fortune mais cela encore sans utiliser de bibliotehque définie uniquement math et matplotlib

“ avec comme explemen notre résultat du diagramme de la phase 1



mais nous avons eu ça comme résultat :



on a donc relancer un prompt ou j'ai encore une fois envoyer l'image en disant ça :

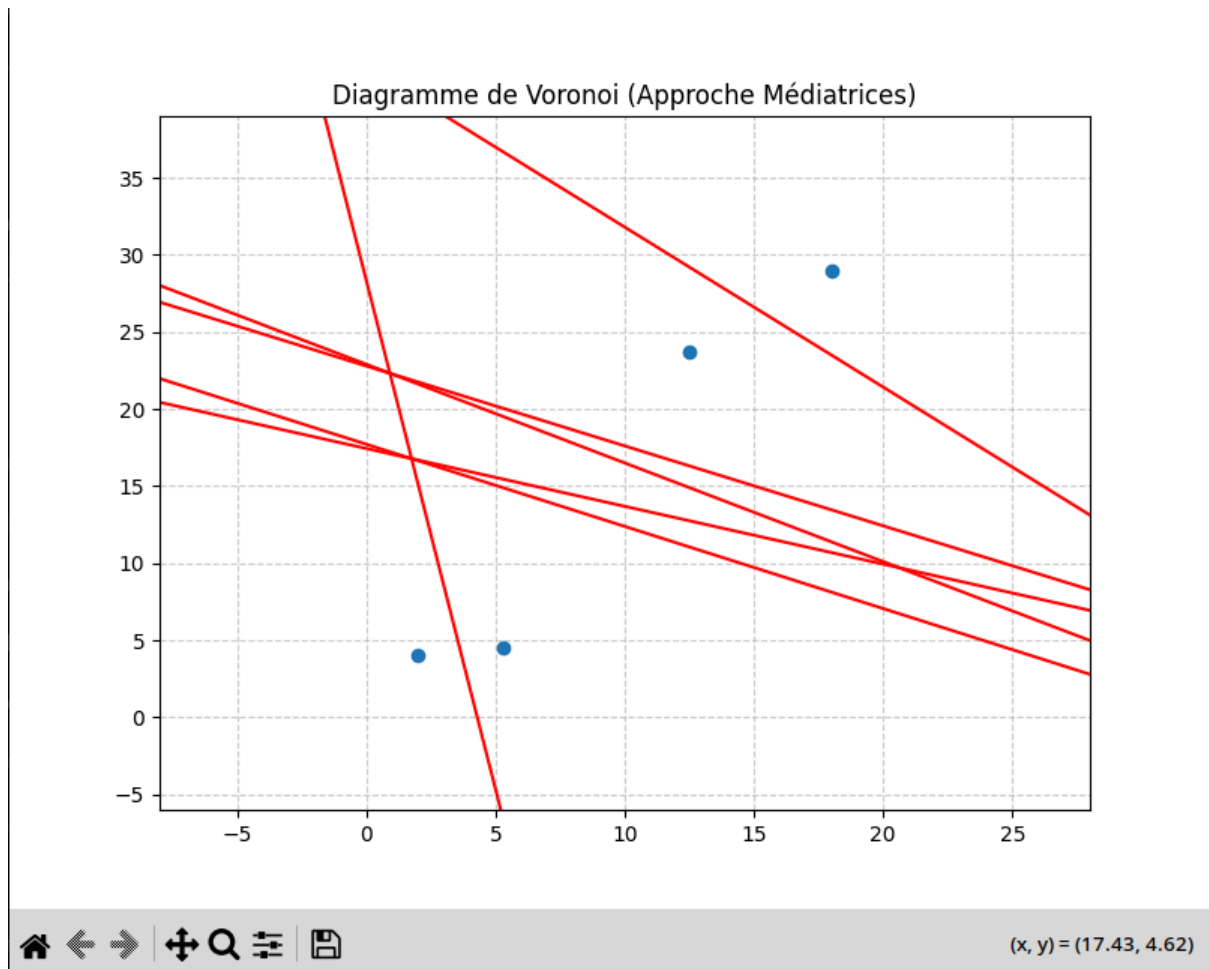
“ce que tu ma rendu ne fait pas du tout un diagramme de voronoi ya juste une seul ligne la je te remontre avec cette exemple de point ce que ça doit rendre 2,4

5.3,4.5

18,29

12.5,23.7”

on encore obtenue un échec



On a envoyé les deux images en comparaison cette fois-ci :

« Tu vois bien qu'entre ces deux images il y a une grande différence. Tu n'as pas bien codé l'algo pour qu'il me donne le résultat de figure_1.png. »

Depuis le début, l'IA essayait une triangulation de Delaunay (via l'algorithme de Bowyer-Watson). Mais après le prompt de comparaison, l'IA a décidé d'implémenter l'algorithme de Fortune.

Suite à ça, l'IA m'a donné un code qui reproduisait bien le diagramme, sauf qu'il le faisait uniquement avec les points que l'on avait donnés en exemple.

Prompt :

“ cela ma bien retourner le bon digramme mais je tai donner des point en exmple si jen rajoute ou je les changes pk a tu fait en sorte que sa me donne touours le meme selon des point que je tavais donner moi je tai dit de lire un fichier donc se fichier peut avoir des point au hasarad “

On a ensuite demandé de nous faire une bonne architecture avec une refactorisation, un changement de nom des variables, des tests, etc., afin d'avoir le plus possible un code avec les bonnes pratiques de programmation, car malgré le premier prompt que l'on lui avait fourni, il avait tout développé dans un seul fichier avec des noms de variables non compréhensibles.

Premier code fournie:

```
import matplotlib.pyplot as plt
```

```
import math
```

```
class Point:
```

```
    def __init__(self, x, y):
```

```
        self.x = x
```

```
        self.y = y
```

```
    def dist_sq(self, other):
```

```
        return (self.x - other.x)**2 + (self.y - other.y)**2
```

```
class Triangle:
```

```
    def __init__(self, p1, p2, p3):
```

```
        self.pts = [p1, p2, p3]
```

```
        # Calcul du centre du cercle circonscrit (circumcenter)
```

```
        self.center, self.radius_sq = self._compute_circumcircle()
```

```
    def _compute_circumcircle(self):
```

```
        p1, p2, p3 = self.pts
```

```
        d = 2 * (p1.x * (p2.y - p3.y) + p2.x * (p3.y - p1.y) + p3.x * (p1.y - p2.y))
```

```
        if abs(d) < 1e-9: return Point(0,0), 0
```

```
        ux = ((p1.x**2 + p1.y**2) * (p2.y - p3.y) + (p2.x**2 + p2.y**2) * (p3.y - p1.y) + (p3.x**2 + p3.y**2) * (p1.y - p2.y)) / d
```

```
        uy = ((p1.x**2 + p1.y**2) * (p3.x - p2.x) + (p2.x**2 + p2.y**2) * (p1.x - p3.x) + (p3.x**2 + p3.y**2) * (p2.x - p1.x)) /
```

```
        center = Point(ux, uy)
```

```
        return center, center.dist_sq(p1)
```

```
    def contains_in_circumcircle(self, p):
```

```
        return p.dist_sq(self.center) < self.radius_sq
```

```

def get_voronoi_edges(triangles):
    """Le Voronoi relie les centres des cercles circonscrits des triangles adjacents."""
    edges = []
    for i in range(len(triangles)):
        for j in range(i + 1, len(triangles)):
            t1, t2 = triangles[i], triangles[j]

            # Si deux triangles partagent deux points, ils sont adjacents
            shared = 0

            for p1 in t1.pts:
                for p2 in t2.pts:
                    if p1 == p2: shared += 1

            if shared == 2:
                edges.append((t1.center, t2.center))

    return edges

```

```

def delaunay_bowyer_watson(points):
    """Algorithme de triangulation de Delaunay."""
    # Création d'un super-triangle qui englobe tous les points
    super_triangle = Triangle(Point(-100, -100), Point(200, -100), Point(50, 200))
    triangulation = [super_triangle]

    for p in points:
        bad_triangles = [t for t in triangulation if t.contains_in_circumcircle(p)]
        polygon = []
        for t in bad_triangles:

```

```

for i in range(3):
    edge = (t.pts[i], t.pts[(i+1)%3])
    is_shared = False
    for other in bad_triangles:
        if other == t: continue
        for j in range(3):
            other_edge = (other.pts[j], other.pts[(j+1)%3])
            if (edge[0] == other_edge[0] and edge[1] == other_edge[1]) or \
                (edge[0] == other_edge[1] and edge[1] == other_edge[0]):
                is_shared = True
        if not is_shared: polygon.append(edge)

for t in bad_triangles: triangulation.remove(t)
for edge in polygon: triangulation.append(Triangle(edge[0], edge[1], p))

return [t for t in triangulation if not any(p in super_triangle.pts for p in t.pts)]

```

```

def read_points(filename):
    points = []
    with open(filename, 'r') as f:
        for line in f:
            if line.strip():
                x, y = map(float, line.replace(',', ' ').split())
                points.append(Point(x, y))
    return points

```

```

def plot_voronoi(points, edges):
    plt.figure(figsize=(8, 6))

    px = [p.x for p in points]; py = [p.y for p in points]

    plt.scatter(px, py, color='blue', label='Points')

    for e in edges:
        plt.plot([e[0].x, e[1].x], [e[0].y, e[1].y], 'r-')

    plt.title("Diagramme de Voronoi (Dual de Delaunay)")

    plt.show()

```

--- TESTS UNITAIRES ---

```

def test_circumcircle():
    t = Triangle(Point(0,0), Point(2,0), Point(1,1))

    assert t.center.x == 1.0 and t.center.y == 0.0

    print("Test géométrique réussi.")

```

```

if __name__ == "__main__":
    # Simulation du fichier texte pour l'exemple

    with open("points.txt", "w") as f:
        f.write("2,4\n5.3,4.5\n18,29\n12.5,23.7")

    test_circumcircle()

    pts = read_points("points.txt")

    triangles = delaunay_bowyer_watson(pts)

    voronoi_edges = get_voronoi_edges(triangles)

```

```
plot_voronoi(pts, voronoi_edges)
```