

SAÉ S6

Phase 2 : Développement assisté par l'IA (Copilot)

Documentation – Prompts et journal de corrections

Équipe P, groupe Makémaké
01/03/2026

Table des matières

1. Prompts utilisés.....	2
A. Première génération de l'IA	2
B. Correction de la première génération de l'IA.....	2
C. Correction de la qualité du code et refactorisation.....	2
D. Problème d'affichage	3
 2. Journal des corrections	 3
A. Cahier des charges pas respecté	3
B. Points codés directement dans le fichier	4
C. Lecture de fichier « fragile »	5
D. Échelle fixe dans l'affichage	5

1. Prompts utilisés

A. Première génération de l'IA

Le premier problème apparu à la première génération du code. On a demandé à Copilot :

« Génère un programme Python qui affiche un diagramme de Voronoï à partir d'une liste de points. » L'IA n'a pas respecté les consignes données car il a utilisé « scipy » et « Voronoi » déjà implémenté. On nous a imposé d'implémenter nous même les algorithmes, donc cela ne pouvait pas fonctionner.

B. Correction de la première génération de l'IA

On a donc demandé à Copilot :

- « Non, il faut coder Delaunay et Voronoï nous-mêmes. »
- « Refais le programme sans aucune bibliothèque externe. »

Il nous a donc proposé d'utiliser l'algorithme de Bowyer Watson, qui est une méthode qui va nous permettre de construire la triangulation de Delaunay directement.

C. Correction de la qualité du code et refactorisation

On a ensuite demandé :

- « Réécris le code proprement en plusieurs fichiers. »
- « Respecte les bonnes pratiques et le clean code. »
- « Ne change pas le comportement, seulement la structure. »

```
voronoi/
|
|-- voronoi_app/
|   |-- __init__.py
|   |-- cli.py
|   |-- io_utils.py
|   |-- delaunay.py
|   |-- geometry.py
|   |-- voronoi.py
|   |-- svg_export.py
|
|-- tests/
|   |-- test_io_utils.py
|   |-- test_geometry.py
|   |-- test_delaunay.py
|   |-- test_voronoi.py
|
|-- points.txt
|-- pytest.ini
|-- requirements.txt
```

D. Problème d’affichage

Ensuite, on a eu des problèmes d’affichages du diagramme de Voronoi :

- « L’échelle ne va pas, les points doivent rester visibles. »
- « Je veux une échelle automatique. »
- « Je veux des graduations régulières : X par 2, Y par 5. »
- « Ajoute une grille. »

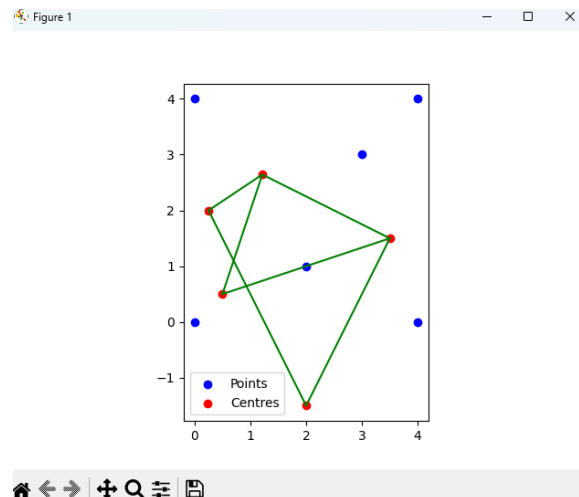


Diagramme de Voronoi (initial)

2. Journal des corrections

A. Cahier des charges pas respecté

Voici le code généré par l’IA :

```
from scipy.spatial import Voronoi, voronoi_plot_2d

points = [...]
vor = Voronoi(points)
voronoi_plot_2d(vor)
plt.show()
```

Donc on peut constater la présence de la bibliothèque « scipy » qui fournit directement l’algorithme de Delaunay, on n’aurait pas eu de calculs.

Voici la correction apportée avec l'implémentation manuelle de Bowyer-Watson et la construction du Voronoi :

```
def bowyer_watson(points):
    triangles = [super_triangle]
    for p in points:
        bad = [t for t in triangles if point_in_circumcircle(p, t)]
        ...
        triangles.append((a, b, p))
    return final_triangles

def build_voronoi(triangles):
    centers = [circumcircle(t)[0] for t in triangles]
    edges = [...]
    return centers, edges
```

B. Points codés directement dans le fichier

Initialement, les points étaient codés en durs dans un fichier qui contenait d'autres responsabilités. Par exemple :

```
points = [(0,0), (4,0), (4,4)]
```

Les consignes exigeaient qu'il fallût créer un fichier en renseignant une liste de points ligne par ligne, donc ça ne respectait pas la consigne mais il y a aussi le côté des bonnes pratiques de code. On mélange la logique métier avec les algorithmes de Delaunay et Voronoi et l'interface utilisateur.

On a donc créé un fichier « io_utils » avec une fonction qui permet de charger une liste de points depuis un fichier texte :

```
def load_points_from_file(filename: str) -> List[Point]:
    points: List[Point] = []
    with open(filename, "r") as f:
        ...
    return points
```

Puis nous avons appelé la fonction dans le fichier cli.py :

```
points = load_points_from_file("points.txt")
```

C. Lecture de fichier « fragile »

Voici le code initial :

```
x, y = line.split(",")
```

Voici le code après correction

```
line = line.replace(",", " ")  
x, y = map(float, line.split())
```

Avant, on utilisait `line.split(",")`, ce qui faisait planter le programme dès qu'une ligne n'était pas strictement au format `x,y`. Après, on remplace la virgule par un espace puis on découpe la ligne (`line.replace(",", " ").split()`), ce qui permet d'accepter `x,y`, `x y`, `x, y` et d'éviter les crashes.

D. Échelle fixe dans l'affichage

Pour le code initial, on avait une échelle fixe donc les points qui étaient situés hors de la zone d'échelle étaient invisibles. Donc on a modifié le code pour pouvoir obtenir une échelle automatique.

Avant :

```
ax.set_xlim(0, 5)  
ax.set_ylim(0, 5)
```

Après :

```
ax.set_xlim(min(xs) - 1, max(xs) + 1)  
ax.set_ylim(min(ys) - 1, max(ys) + 1)
```

Voici notre diagramme de Voronoï final avec Copilot :

