

# Optimisation d'un réseau de caméras de surveillance

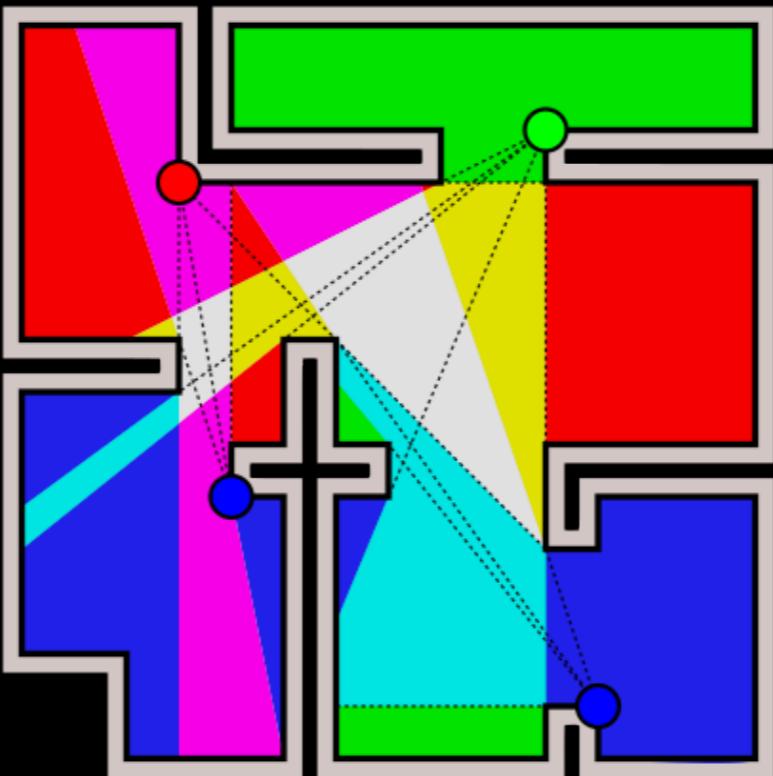
Audic Clément - TIPE 2022-2023 - n° 12345

# Sommaire

1. Le problème de la galerie d'art
2. Le lien avec la triangulation
3. La généralisation du problème à la ville
4. La résolution algorithmique
5. L'application à la ville de Besançon

# Problème de la galerie d'art

“ Quelle est le minimum de caméras nécessaires pour surveiller toute une galerie d'art ? ”

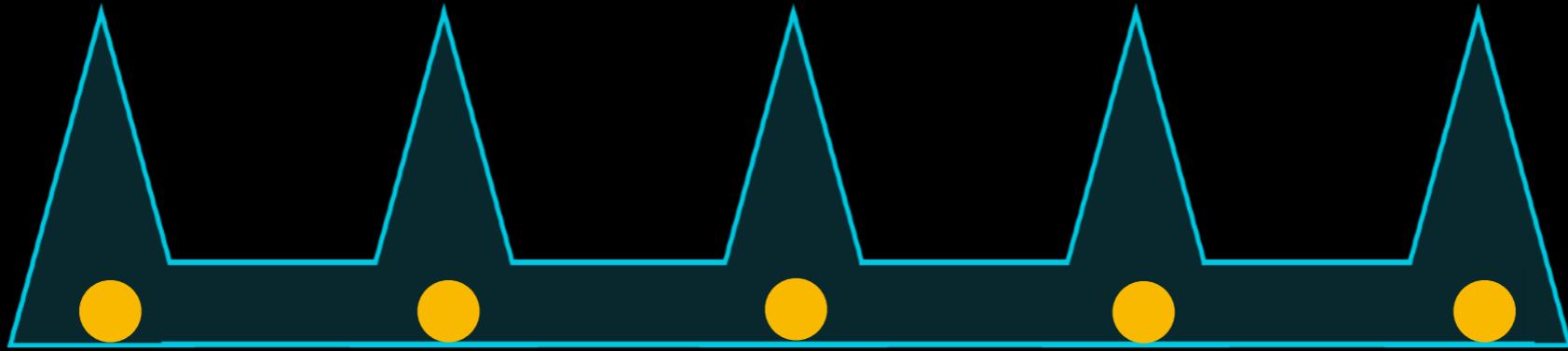


[https://en.wikipedia.org/wiki/Art\\_gallery\\_problem](https://en.wikipedia.org/wiki/Art_gallery_problem)

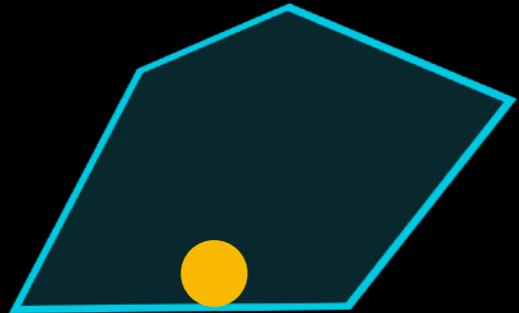
# Théorème de la galerie d'art (Chvátal)

Tout polygone simple à  $n$  côtés peut être gardé par au plus  $\lfloor n/3 \rfloor$  gardes.

Pire cas:

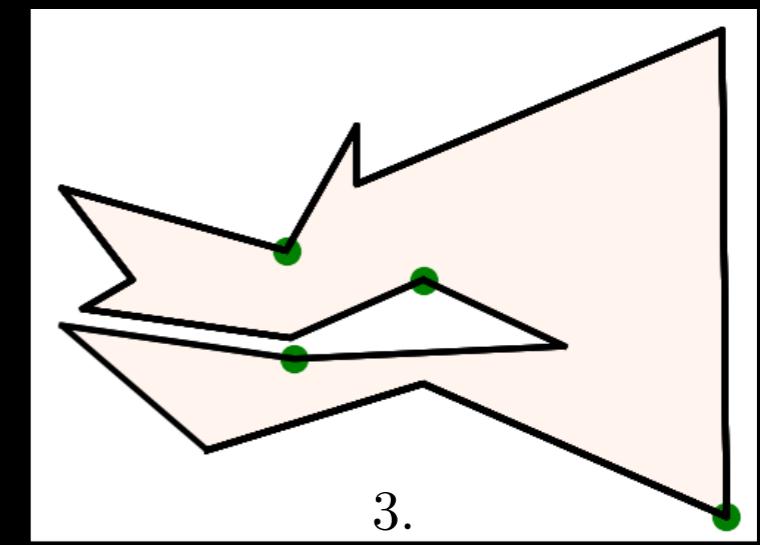
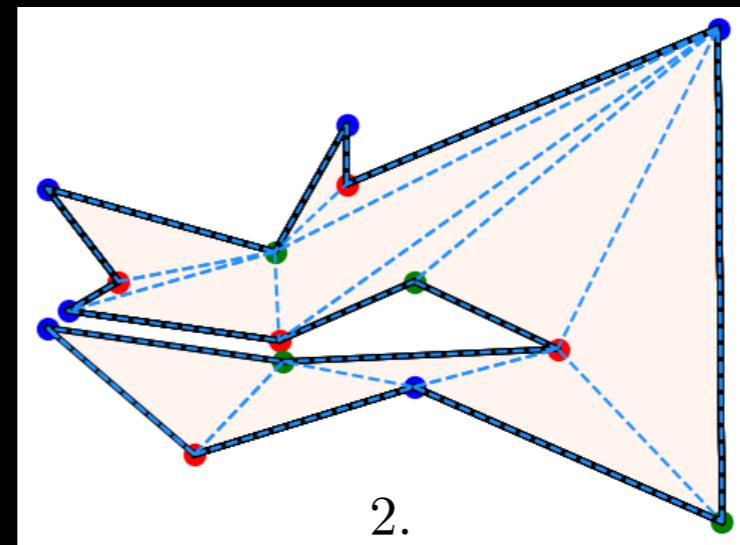
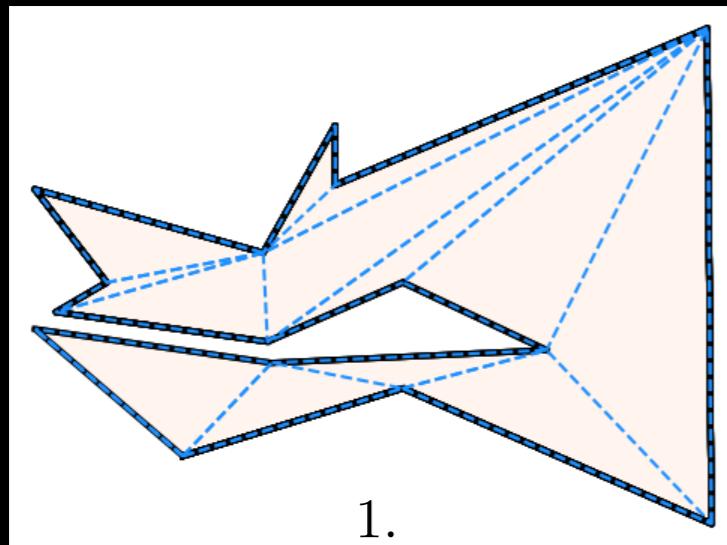
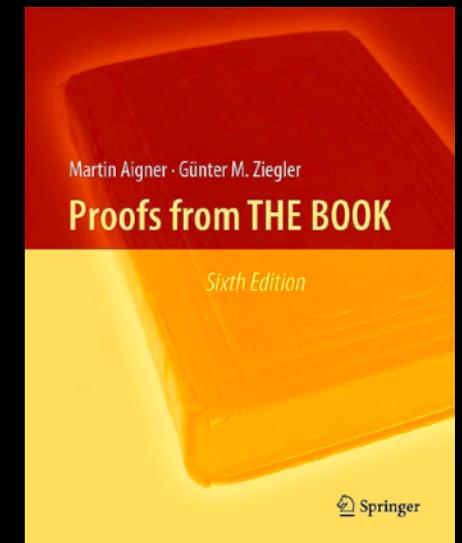


Meilleur cas:



# Preuve constructiviste par triangulation (Fisk)

1. Triangulation (Tout polygone peut être triangulé)
2. 3-coloriage (Toute triangulation peut être 3-coloriée)
3. Choix du plus petit groupe de couleurs

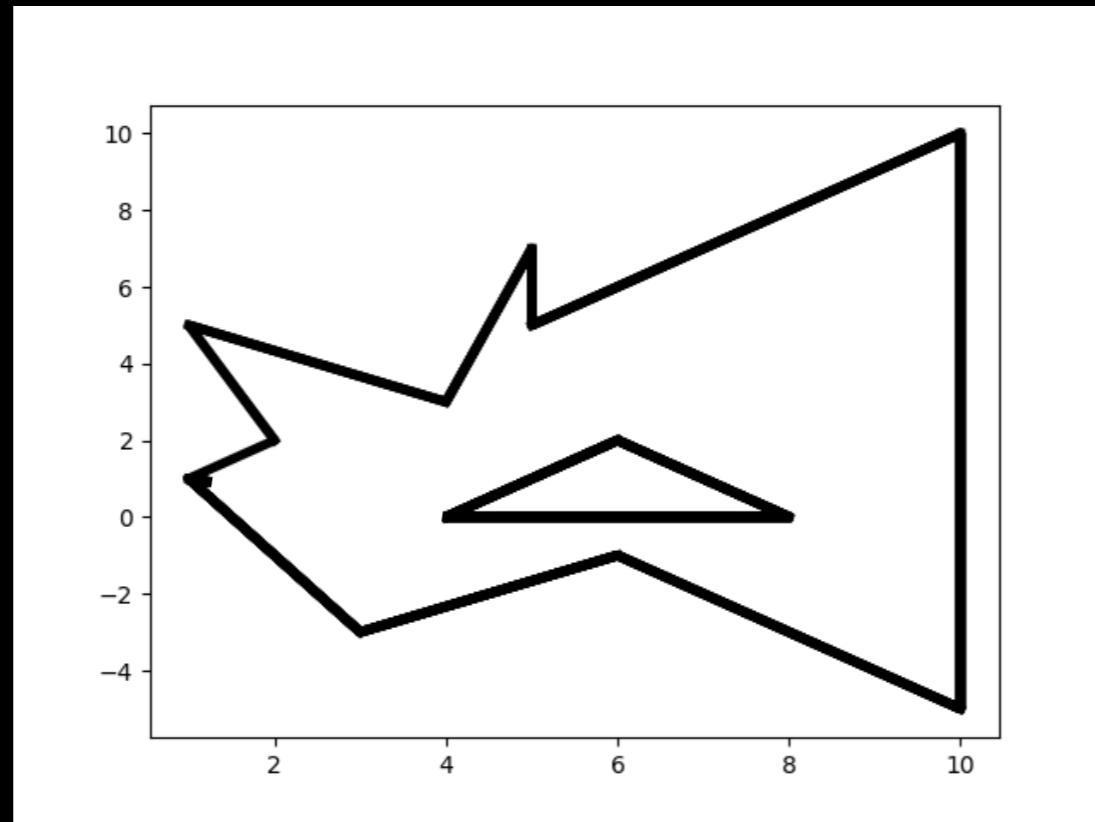


$$\textcolor{red}{R} + \textcolor{blue}{B} + \textcolor{green}{V} = n$$

$$\textcolor{red}{R} \leq \lfloor n/3 \rfloor \text{ ou } \textcolor{blue}{B} \leq \lfloor n/3 \rfloor \text{ ou } \textcolor{green}{V} \leq \lfloor n/3 \rfloor$$

# Adaptation du problème pour une ville

Tout polygone à  $n$  côtés et  $h$  trous peut être surveillé par au plus  $\lfloor(n + h)/3\rfloor$  caméras.

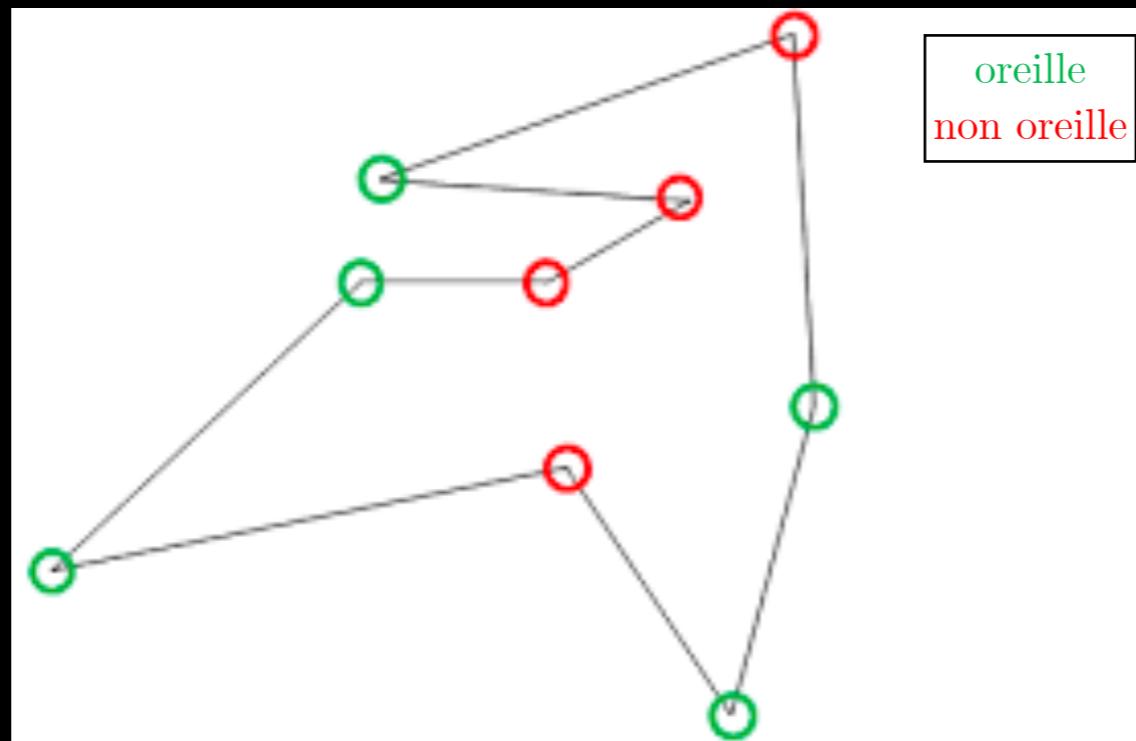


# Résolution algorithmique

## 1) Triangulation: algorithme de ear clipping

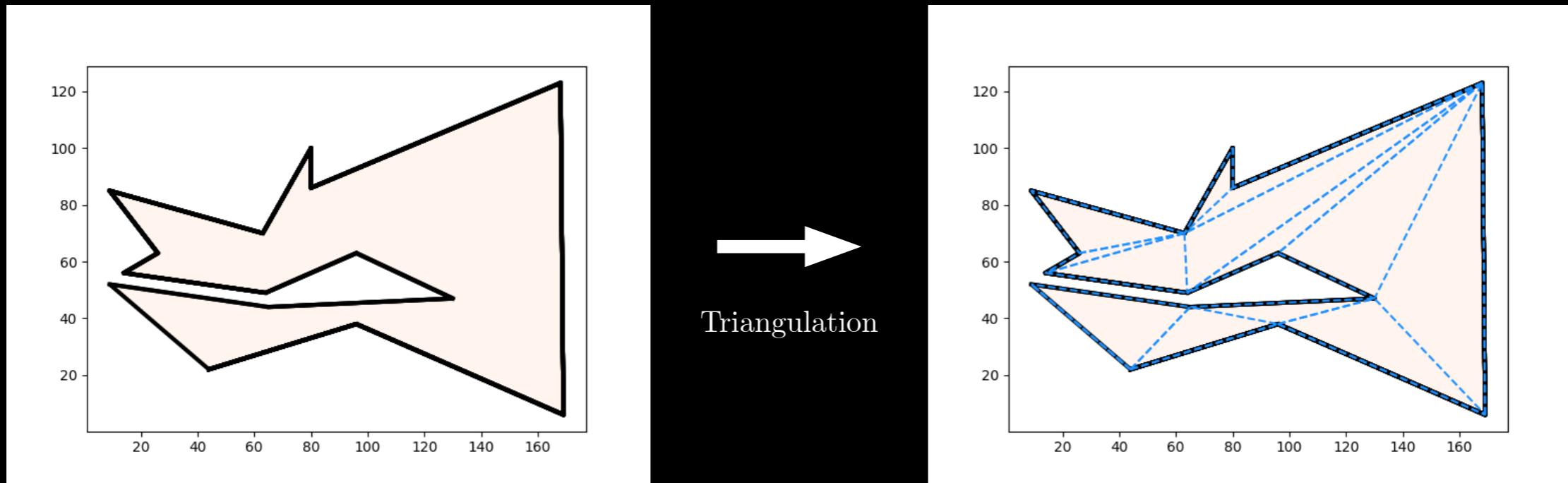
Définition oreille:

- angle saillant
- le triangle formé avec ses deux sommets adjacents ne contient aucun point



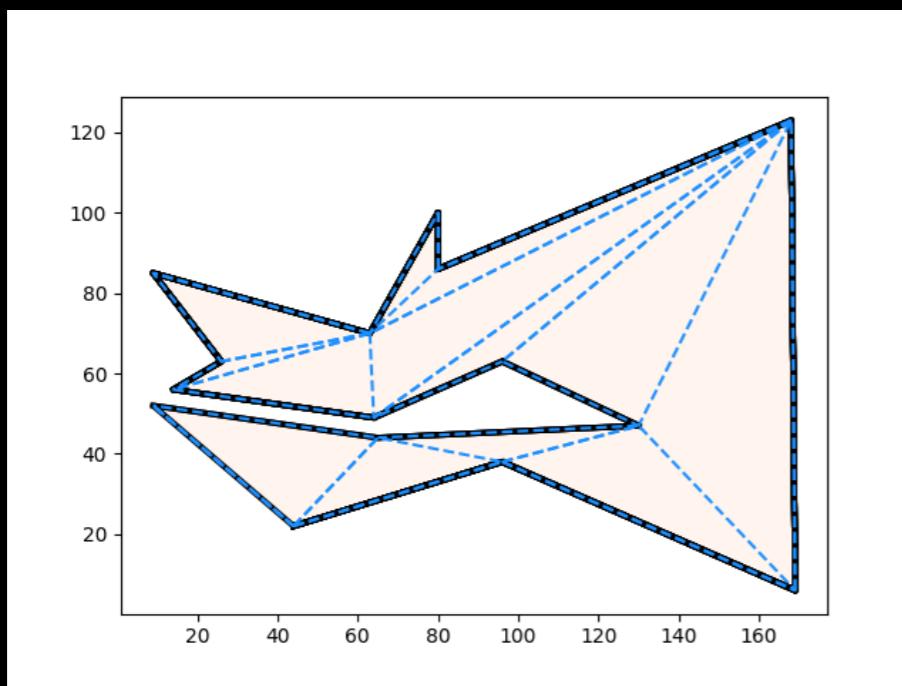
<http://www.sunshine2k.de/coding/java/Polygon/Kong/Kong.html>

## Exemple d'exécution de mon implémentation

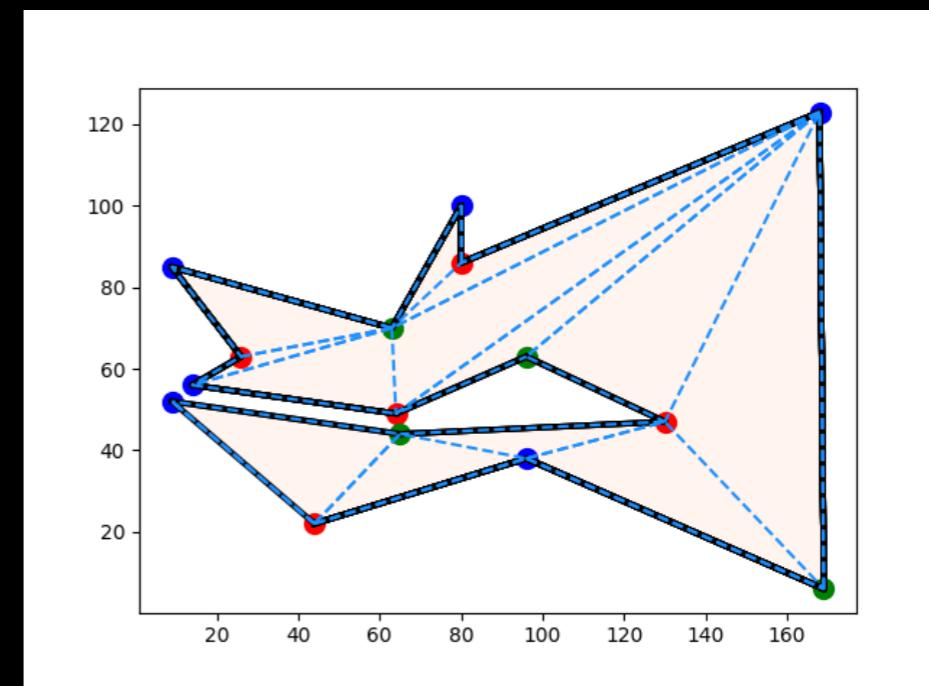


## 2) 3-coloriage: algorithme de proche en proche

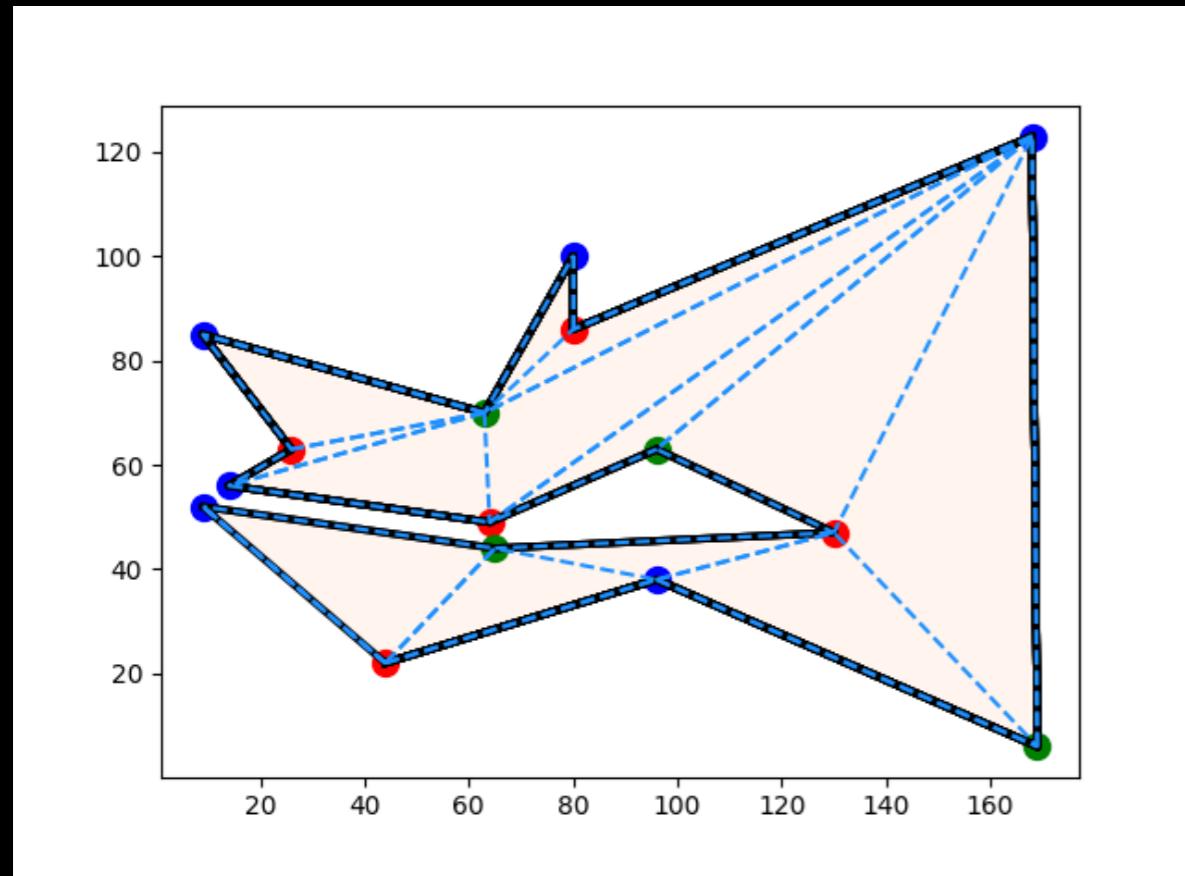
Exemple d'exécution de mon implémentation



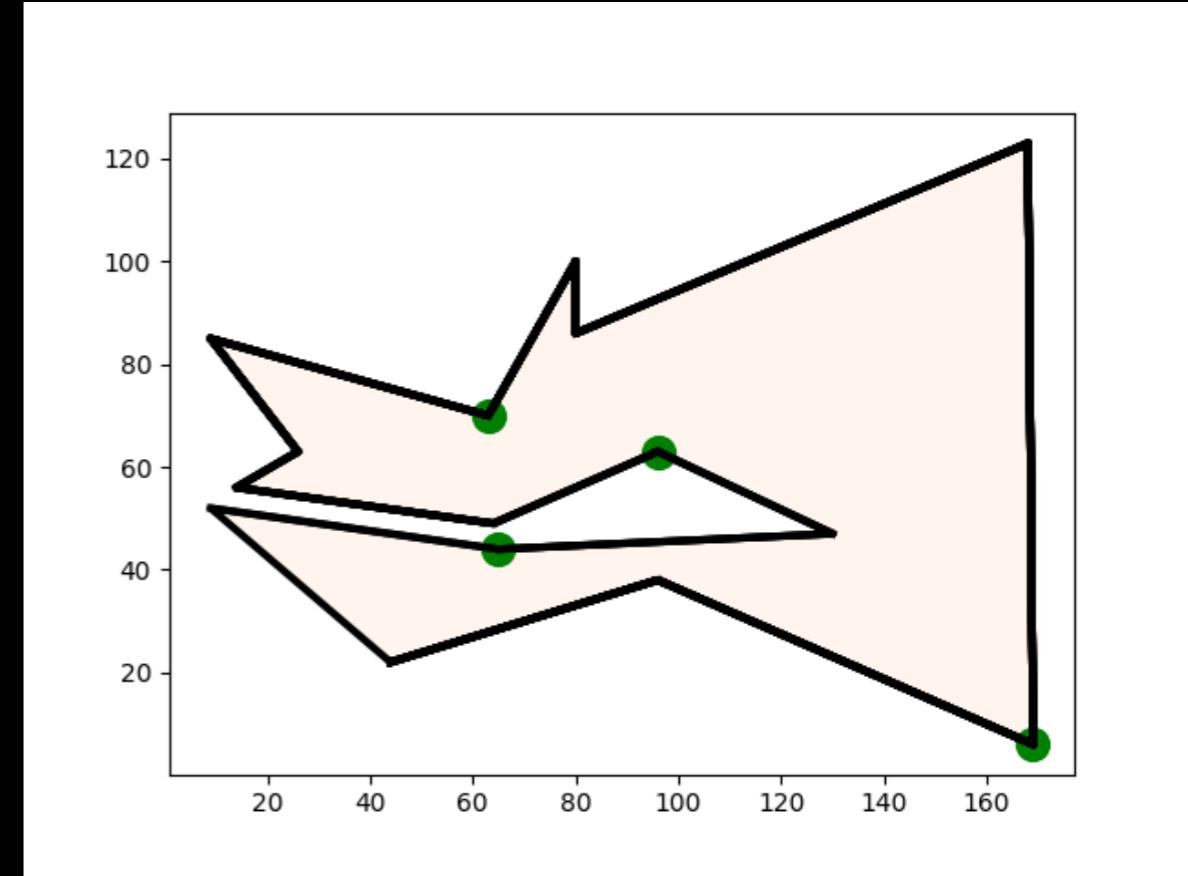
3-coloration



### 3) Affichage



`affichage_global(polygone)`

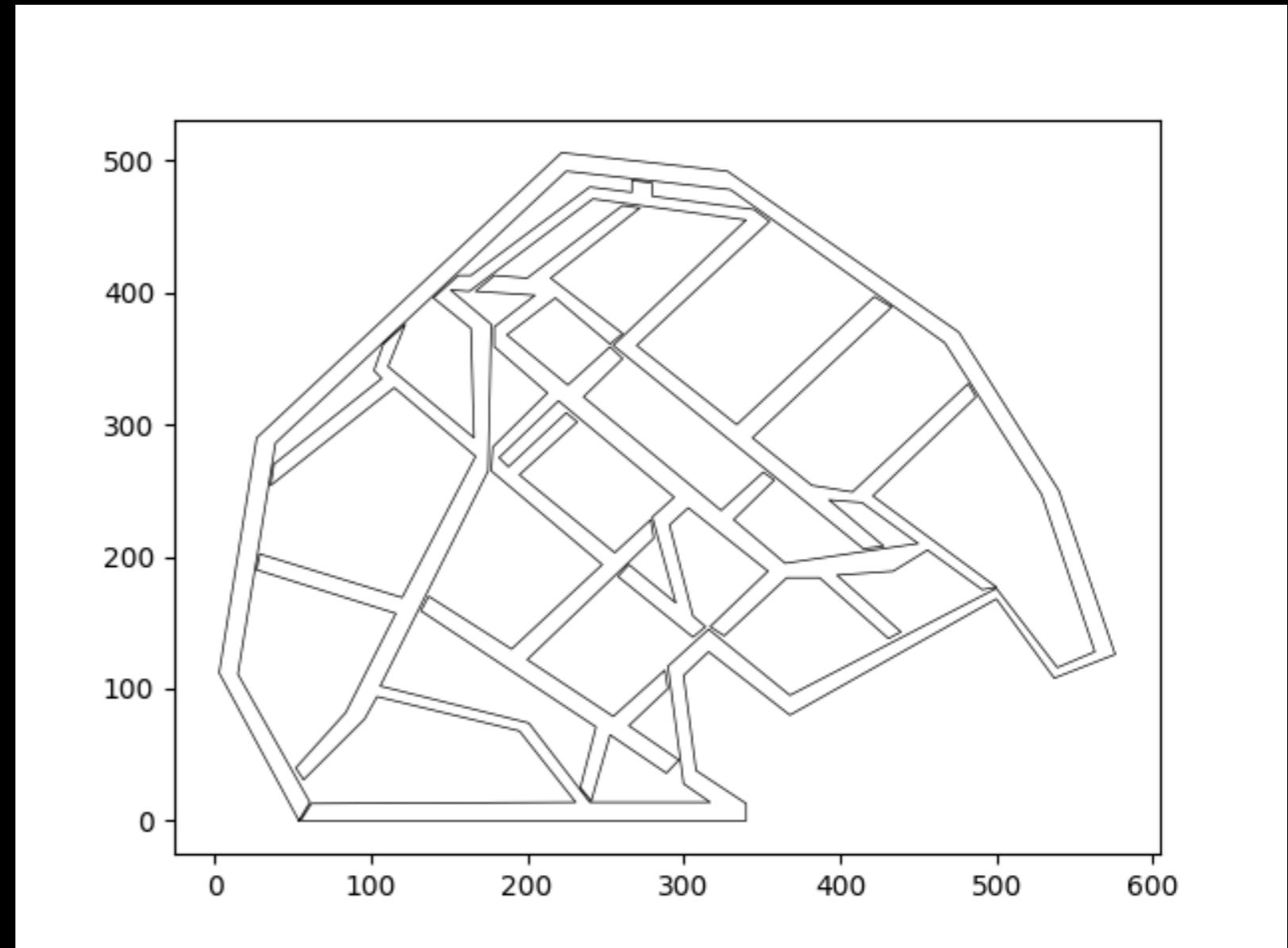
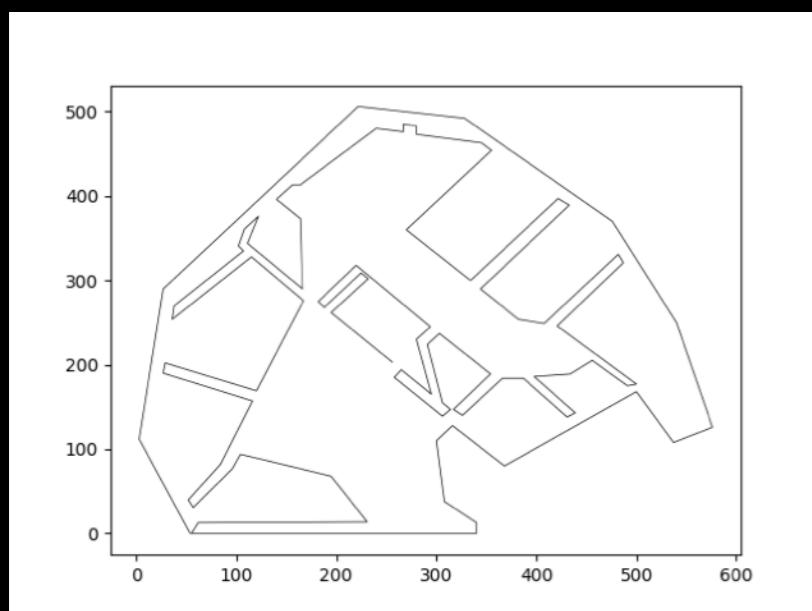
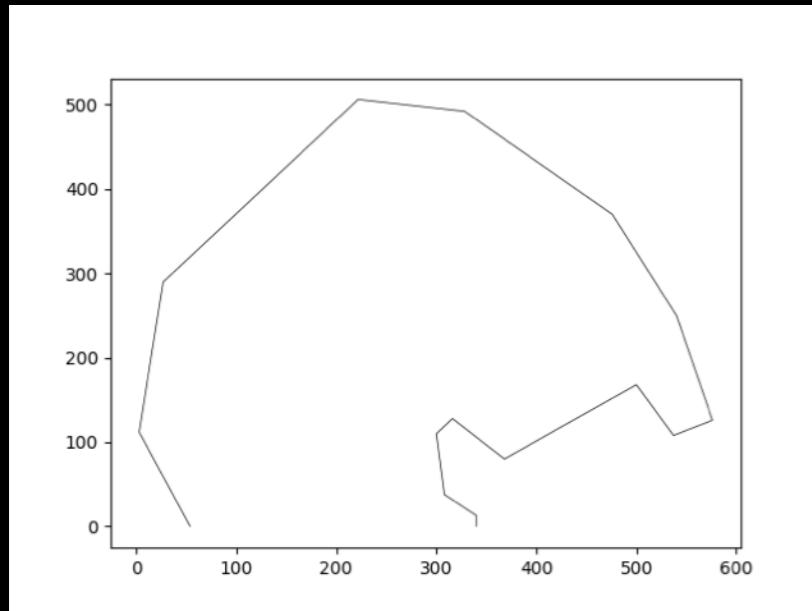


`affichage_solution(polygone)`

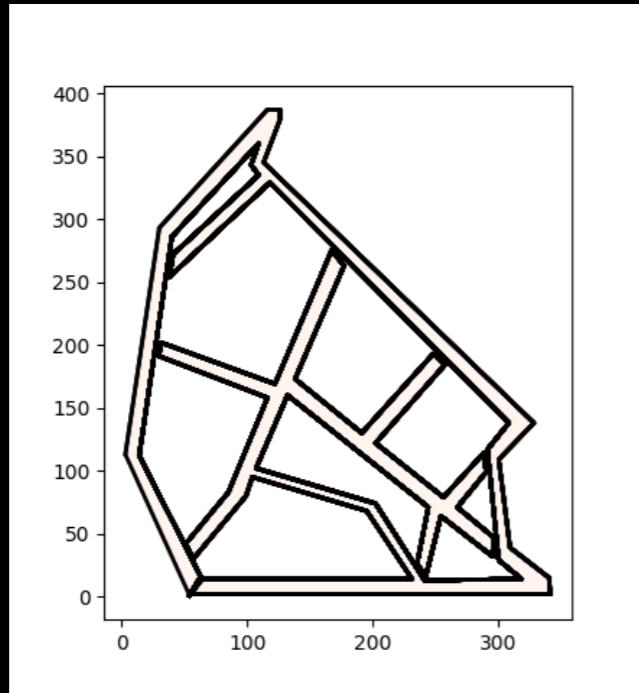
# Application pour la ville de Besançon



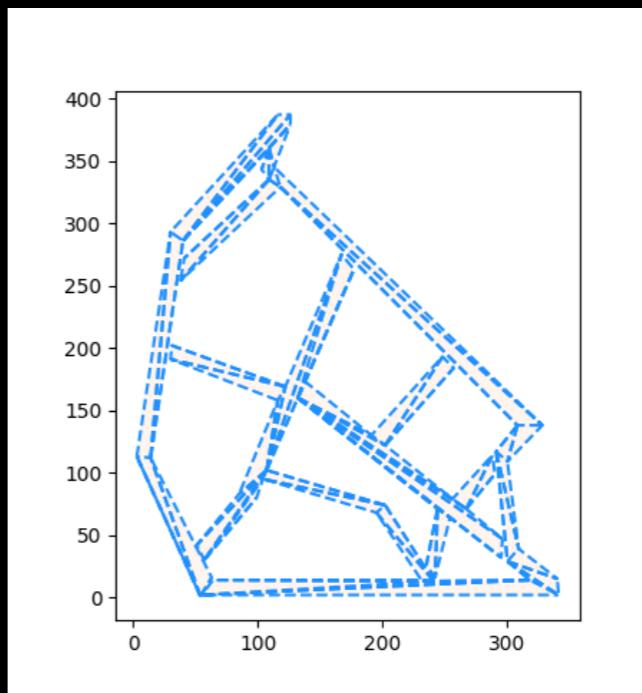
# Utilisation de GeoGebra



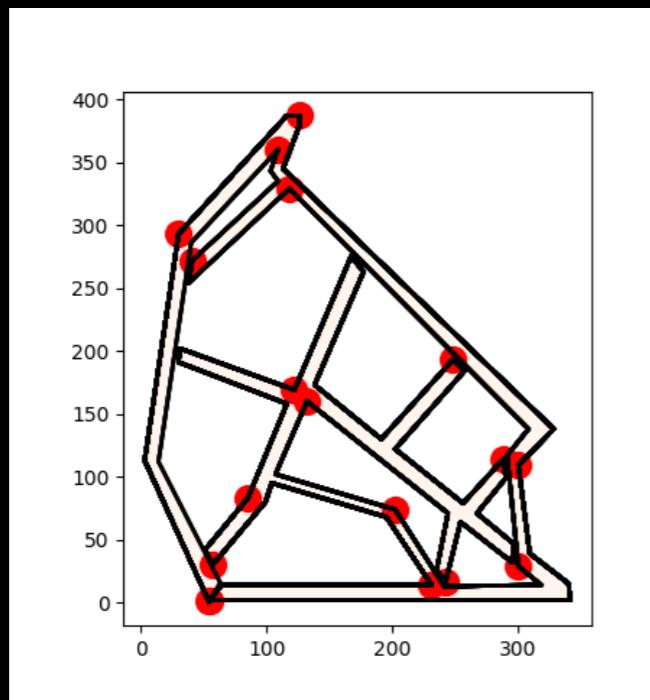
# Détails de l'exécution



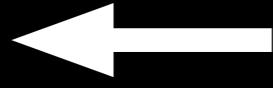
Triangulation



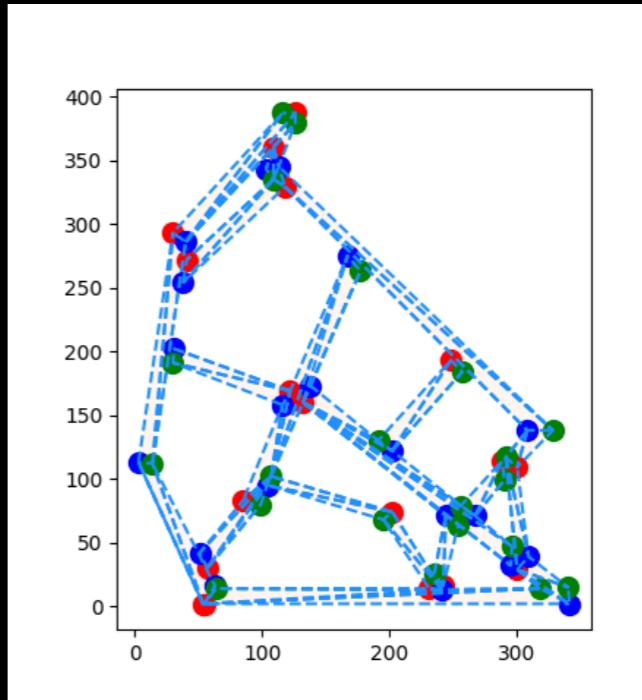
3-coloration



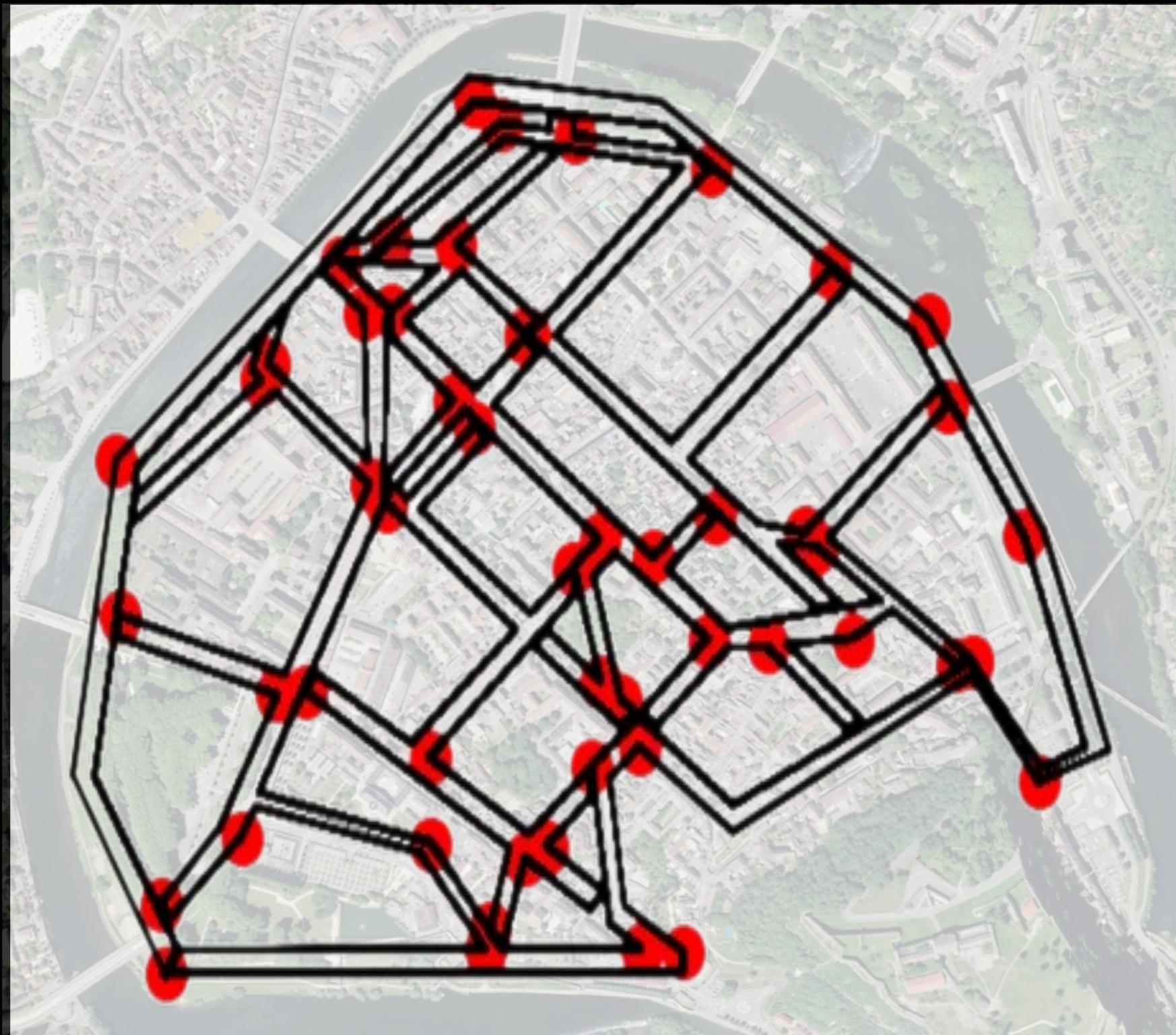
Choix de la couleur



14



# Résultat final



# Annexes: code

```
● ● ●

1  from matplotlib import pyplot
2  from shapely.geometry import Polygon
3  import geopandas
4  import sys
5
6  '''partie 1: triangulation'''
7
8  def ear_clipping(polygone):
9
10     if test_sens_horaire(polygone):
11         polygone.reverse()
12
13     nb_points = len(polygone)
14     liste_indices = list(range(nb_points))
15     liste_triangles = []
16
17     while(nb_points>3):
18         for indice_point in liste_indices:
19             indice_indice_point = liste_indices.index(indice_point)
20             point_prec = polygone[liste_indices[indice_indice_point-1]]
21             point = polygone[indice_point]
22             point_suiv = polygone[liste_indices[(indice_indice_point + 1) % nb_points]]
23
24             if test_oreille(point_prec, point, point_suiv, polygone):
25                 liste_triangles.append((point_prec, point, point_suiv))
26                 del liste_indices[indice_indice_point]
27                 nb_points -= 1
28                 break
29
30     liste_triangles.append((polygone[liste_indices[0]], polygone[liste_indices[1]], polygone[liste_indices[2]]))
31
32     return liste_triangles
33
34 # l'algorithme de ear clipping doit toujour traiter les sommets du polygone dans le même sens, ici, le sens anti horaire a été choisi arbitrairement
35 def test_sens_horaire(polygone):
36     aire_polygone = 0
37     taille_polygone = len(polygone)
38     for i in range(taille_polygone):
39         point = polygone[i]
40         point_suiv = polygone[(i + 1) % taille_polygone]
41         aire_polygone += (point_suiv[0] - point[0]) * ((point_suiv[1] + point[1])/2)
42     return aire_polygone > 0
43
44 # deux conditions pour qu'un sommet soit une oreille d'un polygone
45 def test_oreille(p1, p2, p3, polygone):
46     booleen = test_saillant(p1, p2, p3) and test_triangle_contient_points(p1, p2, p3, polygone)
47     return booleen
48
```

```

48
49 # condition n°1: les deux arêtes du sommet forme un angle saillant
50 def test_saillant(prec, point, suiv):
51     return aux_test_saillant(prec[0], prec[1], point[0], point[1], suiv[0], suiv[1]) < 0
52
53 def aux_test_saillant(x1, y1, x2, y2, x3, y3):
54     return x1 * (y3 - y2) + x2 * (y1 - y3) + x3 * (y2 - y1)
55
56 # condition n°2: le triangle formé par le sommet et ces deux sommets voisins ne contient aucun autres sommets du polygone
57 def test_triangle_contient_points(p1, p2, p3, polygone):
58     for pi in polygone:
59         if pi in (p1, p2, p3):
60             continue
61         elif test_point_interieur_triangle(pi, p1, p2, p3):
62             return False
63     return True
64
65 def test_point_interieur_triangle(p, a, b, c):
66     aire = aire_triangle(a[0], a[1], b[0], b[1], c[0], c[1])
67     aire1 = aire_triangle(p[0], p[1], a[0], a[1], b[0], b[1])
68     aire2 = aire_triangle(p[0], p[1], b[0], b[1], c[0], c[1])
69     aire3 = aire_triangle(p[0], p[1], a[0], a[1], c[0], c[1])
70     boolean_test_egalite_aire = abs(aire - (aire1 + aire2 + aire3)) < sys.float_info.epsilon
71     return boolean_test_egalite_aire
72
73 def aire_triangle(x1, y1, x2, y2, x3, y3):
74     return abs((x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2))) / 2.0
75
76 def calcul_adjacence(triangles):
77     dictionnaire_adjacences = {}
78     for triangle in triangles:
79         compteur = 0
80         for point in triangle:
81             if point not in dictionnaire_adjacences:
82                 dictionnaire_adjacences[point] = [triangle[(compteur + 1) % 3], triangle[(compteur + 2) % 3]]
83             else:
84                 dictionnaire_adjacences[point] += [triangle[(compteur + 1) % 3], triangle[(compteur + 2) % 3]]
85             compteur += 1
86
87     for point in dictionnaire_adjacences.keys():
88         dictionnaire_adjacences[point] = list(set(dictionnaire_adjacences[point]))
89
90     return dictionnaire_adjacences
91
92 """partie 2: triangulation"""
93
94 def trois_coloriage(triangles):
95     liste_colories = []
96     dictionnaire_adjacences = calcul_adjacence(triangles)
97     n = len(dictionnaire_adjacences)
98     dictionnaire_couleurs = {}
99

```

```

100     # coloriage du premier triangle
101     triangle = triangles[0]
102     premier = triangle[0]
103     deuxieme = triangle[1]
104     troisieme = triangle[2]
105     dictionnaire_couleurs[premier] = 1
106     dictionnaire_couleurs[deuxieme] = 2
107     dictionnaire_couleurs[troisieme] = 3
108     liste_colories.append(premier)
109     liste_colories.append(deuxieme)
110     liste_colories.append(troisieme)
111     liste_attente = []
112     if(n!=3):
113         coloration_rec(liste_attente,premier,deuxieme,troisieme,liste_colories,n,dictionnaire_adjacences,dictionnaire_couleurs)
114     liste_rouge = [cle for cle, valeur in dictionnaire_couleurs.items() if valeur == 1]
115     liste_bleu = [cle for cle, valeur in dictionnaire_couleurs.items() if valeur == 2]
116     liste_vert = [cle for cle, valeur in dictionnaire_couleurs.items() if valeur == 3]
117     return [liste_rouge,liste_bleu,liste_vert]
118
119 def coloration_rec(liste_attente,premier,deuxieme,troisieme,liste_colories,nb_sommets,dictionnaire_adjacences,dictionnaire_couleurs):
120
121     while len(list(set(liste_colories)))!=nb_sommets:
122
123         listea = test_coloration(dictionnaire_adjacences,deuxieme,troisieme,liste_colories)
124         listeb = test_coloration(dictionnaire_adjacences,premier,troisieme,liste_colories)
125         listec = test_coloration(dictionnaire_adjacences,premier,deuxieme,liste_colories)
126
127         if listea[0] and listeb[0]:
128             liste_attente.append((listeb[1],premier,troisieme))
129         elif listea[0] and listec[0]:
130             liste_attente.append((listec[1],premier,deuxieme))
131         elif listeb[0] and listec[0]:
132             liste_attente.append((listec[1],premier,deuxieme))
133
134         if listea[0]:
135             coloriage(listea[1],dictionnaire_couleurs[deuxieme],dictionnaire_couleurs[troisieme],dictionnaire_couleurs,liste_colories)
136             liste_colories.append(listea[1])
137             return coloration_rec(liste_attente,deuxieme,troisieme,listea[1],liste_colories,nb_sommets,dictionnaire_adjacences,dictionnaire_couleurs)
138
139         if listeb[0]:
140             coloriage(listeb[1],dictionnaire_couleurs[premier],dictionnaire_couleurs[troisieme],dictionnaire_couleurs,liste_colories)
141             liste_colories.append(listeb[1])
142             return coloration_rec(liste_attente,premier,troisieme,listeb[1],liste_colories,nb_sommets,dictionnaire_adjacences,dictionnaire_couleurs)
143
144         if listec[0]:
145             coloriage(listec[1],dictionnaire_couleurs[premier],dictionnaire_couleurs[deuxieme],dictionnaire_couleurs,liste_colories)
146             liste_colories.append(listec[1])
147             return coloration_rec(liste_attente,premier,deuxieme,listec[1],liste_colories,nb_sommets,dictionnaire_adjacences,dictionnaire_couleurs)
148
149         else:
150             if liste_attente is not None:
151                 l = liste_attente[0]
152                 coloriage(l[0],dictionnaire_couleurs[l[1]],dictionnaire_couleurs[l[2]],dictionnaire_couleurs,liste_colories)
153                 liste_colories.append(l[0])
154                 liste_attente_copie = liste_attente
155                 liste_attente_copie.remove(l)
156                 return coloration_rec(liste_attente_copie,l[0],l[1],l[2],liste_colories,nb_sommets,dictionnaire_adjacences,dictionnaire_couleurs)
157

```

```

157
158     def intersection(liste1, liste2):
159         return list(set(liste1) & set(liste2))
160
161     def complementaire(liste1,liste2):
162         return list(set(liste1) - set(liste2))
163
164     def test_coloration (dictionnaire_adjacences,sommet1,sommet2,liste_colories):
165         liste = complementaire(intersection(dictionnaire_adjacences[sommet1],dictionnaire_adjacences[sommet2]),liste_colories)
166         if liste != []:
167             return [True,liste[0]]
168         else:
169             return [False,[42,42,42]]
170
171     def coloriage(elt,couleur1,couleur2,dictionnaire_couleurs,liste_colories):
172         if couleur1 + couleur2 == 3 :
173             dictionnaire_couleurs[elt]= 3
174         elif couleur1 + couleur2 == 5 :
175             dictionnaire_couleurs[elt]= 1
176         else:
177             dictionnaire_couleurs[elt]= 2
178         liste_colories.append(elt)
179
180     '''partie 3: affichage'''
181
182     def affichage_global(polygone):
183         triangles = ear_clipping (polygone)
184         affichage_polygone(polygone)
185         affichage_triangulation(triangles)
186         affichage_trois_coloriage(triangles)
187         pyplot.show()
188
189     def affichage_solution(polygone):
190         affichage_polygone(polygone)
191         triangles = ear_clipping (polygone)
192         affichage_couleur_minimale(triangles)
193         pyplot.show()
194
195     def affichage_polygone(polygone):
196         # légère coloration de l'intérieur du polygone
197         poly = Polygon(polygone)
198         pol = geopandas.GeoSeries(poly)
199         pol.plot(color='seashell')
200         # tracé du contour du polygone
201         liste_x = []
202         liste_y = []
203         for point in polygone:
204             liste_x.append(point[0])
205             liste_y.append(point[1])
206             pyplot.plot(liste_x,liste_y, linewidth=2, c = 'black')
207
208     def affichage_triangulation(triangles):
209         for i in triangles:
210             liste_x = []
211             liste_y = []
212             for j in i:
213                 liste_x.append(j[0])
214                 liste_y.append(j[1])
215                 pyplot.plot(liste_x,liste_y,color='dodgerblue',linestyle='dashed')
216

```

```

216
217     def affichage_trois_coloriage(triangles):
218         liste_couleurs = trois_coloriage(triangles)
219         liste_rouge_x = []
220         liste_rouge_y = []
221         liste_bleu_x = []
222         liste_bleu_y = []
223         liste_vert_x = []
224         liste_vert_y = []
225
226         for point in liste_couleurs[0]:
227             liste_rouge_x.append(point[0])
228             liste_rouge_y.append(point[1])
229         for point in liste_couleurs[1]:
230             liste_bleu_x.append(point[0])
231             liste_bleu_y.append(point[1])
232         for point in liste_couleurs[2]:
233             liste_vert_x.append(point[0])
234             liste_vert_y.append(point[1])
235
236         pyplot.scatter(liste_rouge_x,liste_rouge_y, s=100, c = 'r')
237         pyplot.scatter(liste_bleu_x,liste_bleu_y, s=100, c = 'b')
238         pyplot.scatter( liste_vert_x, liste_vert_y, s=100, c = 'g')
239
240     def affichage_couleur_minimale(triangles):
241         liste_couleurs = trois_coloriage(triangles)
242         liste_rouge_x = []
243         liste_rouge_y = []
244         liste_bleu_x = []
245         liste_bleu_y = []
246         liste_vert_x = []
247         liste_vert_y = []
248
249         taille_liste_rouge = len(liste_couleurs[0])
250         taille_liste_bleu = len(liste_couleurs[1])
251         taille_liste_vert = len(liste_couleurs[2])
252         min_couleurs= min([taille_liste_rouge,taille_liste_bleu,taille_liste_vert])
253
254         if(min_couleurs==taille_liste_rouge):
255             for point in liste_couleurs[0]:
256                 liste_rouge_x.append(point[0])
257                 liste_rouge_y.append(point[1])
258         elif(min_couleurs==taille_liste_bleu):
259             for point in liste_couleurs[1]:
260                 liste_bleu_x.append(point[0])
261                 liste_bleu_y.append(point[1])
262         else:
263             for point in liste_couleurs[2]:
264                 liste_vert_x.append(point[0])
265                 liste_vert_y.append(point[1])
266
267         pyplot.scatter(liste_rouge_x,liste_rouge_y, s=165, c = 'r')
268         pyplot.scatter(liste_bleu_x,liste_bleu_y, s=165, c = 'b')
269         pyplot.scatter( liste_vert_x, liste_vert_y, s=165, c = 'g')
270

```

```

270
271     '''partie 4: exécution'''
272
273     polygone_illustration = [(44,22),(96,38),(169,6),(168,123),(80,86),(80,100),(63,70),(9,85),(26,63),(14,56),(64,49),(96,63),(130,47),(65,44),(9,52),(44,22)]
274
275     polygone_un = [(54,1),(3,113),(30,293),(116,387),(126,387),(126,379),(113,345),(328,138),(300,109),(309,39),(340,15),(341,2),(55,2),(64,14),(231,14),
276     (195,68),(104,95),(99,80),(57,30),(51,42),(85,83),(116,158),(30,191),(31,202),(122,169),(167,275),(177,263),(138,173),(191,130),(248,193),(257,184),
277     (202,122),(256,79),(288,114),(290,99),(267,71),(296,47),(295,32),(254,64),(242,16),(235,26),(244,71),(132,160),(107,102),(202,74),(241,13),(318,14),
278     (300,29),(291,117),(308,138),(118,329),(38,254),(41,272),(109,335),(103,343),(109,360),(40,286),(14,112),(63,16),(54,1)]
279
280     polygone_deux = [(295,246),(257,203),(266,195),(285,216),(296,165),(307,154),(292,225),(313,247),(236,321),(359,459),(282,474),(282,484),(330,478),
281     (343,467),(355,471),(330,490),(223,506),(131,405),(165,372),(168,290),(176,290),(176,377),(153,402),(184,401),(192,412),(179,414),(244,471),(259,468),
282     (262,477),(242,481),(166,414),(157,415),(226,492),(268,487),(266,469),(197,407),(207,400),(182,376),(181,359),(213,326),(222,334),(191,367),(218,395),
283     (251,363),(259,371),(215,411),(276,467),(343,455),(227,329),(224,333),(179,283),(185,279),(220,318),(228,311),(191,270),(198,264),(235,305),(295,246)]
284
285     polygone_trois = [(263,350),(271,358),(333,299),(422,396),(356,453),(363,461),(476,368),(539,247),(575,124),(534,106),(498,164),(367,81),(318,126),
286     (326,134),(367,95),(432,135),(386,182),(323,137),(316,145),(353,187),(315,225),(360,272),(380,251),(407,247),(480,328),(485,319),(420,245),(495,179)
287     ,(488,176),(455,204),(433,190),(401,187),(394,197),(450,210),(413,239),(392,242),(430,205),(418,200),(359,257),(331,227),(363,195),(388,195),(440,140)
288     ,(500,175),(537,116),(561,127),(523,254),(466,361),(431,388),(344,289),(365,275),(349,268),(263,349)]
289
290     besancon = [(54,0),(3,112),(27,290),(222,506),(328,492),(476,370),(540,250),(576,126),(537,108),(500,168),(368,80),(316,128),(300,110),(308,38),
291     (340,13),(340,0),(55,0),(62,13),(231,14),(195,68),(104,94),(96,77),(57,31),(52,40),(84,82),(116,157),(27,190),(29,202),(120,169),(167,276),(115,328),
292     (36,254),(38,270),(107,335),(102,341),(108,361),(122,376),(111,344),(166,290),(164,373),(140,396),(156,413),(164,413),(240,480),(267,476),(267,485),
293     (280,483),(280,473),(345,463),(355,454),(270,360),(334,300),(422,397),(433,389),(344,290),(382,254),(408,249),(482,331),(487,321),(421,246),(500,177),
294     (491,175),(456,205),(434,189),(398,186),(439,143),(431,138),(387,184),(366,184),(326,140),(317,147),(354,189),(303,237),(291,224),(306,155),(314,147),
295     (306,139),(258,185),(265,194),(295,165),(280,230),(294,245),(220,318),(182,275),(188,268),(225,309),(232,302),(195,262),(256,203),(279,228),(281,214),
296     (200,122),(255,79),(288,114),(290,100),(265,72),(297,46),(289,36),(253,65),(241,15),(234,25),(244,71),(132,159),(137,170),(190,130),(248,194),(177,265),
297     (178,283),(213,324),(179,359),(179,374),(205,398),(167,401),(179,413),(200,411),(261,466),(272,464),(215,411),(261,369),(253,361),(218,396),(187,368),
298     (226,330),(253,359),(261,350),(236,321),(324,235),(351,264),(358,258),(332,228),(365,195),(450,210),(414,241),(393,243),(428,208),(415,206),(255,360),
299     (340,455),(242,471),(163,401),(151,402),(177,376),(175,265),(106,102),(201,74),(240,14),(317,14),(300,28),(290,117),(316,145),(368,95),(500,176),
300     (539,116),(563,128),(529,247),(467,362),(330,478),(225,492),(39,286),(15,111),(61,14),(54,0)]
301
302     affichage_global(polygone_illustration)
303
304     affichage_solution(besancon)
305

```