REY Alice

Numéro de candidat : 25550

# Diminuer le temps de trajet des médecins en visite à domicile

## Modélisation par le problème du voyageur de commerce

https://www.librairie-voyage.com/i-g-n-carte-au-1-25-000eme-serie-bleue-2535sb-saint-flour-murat.html

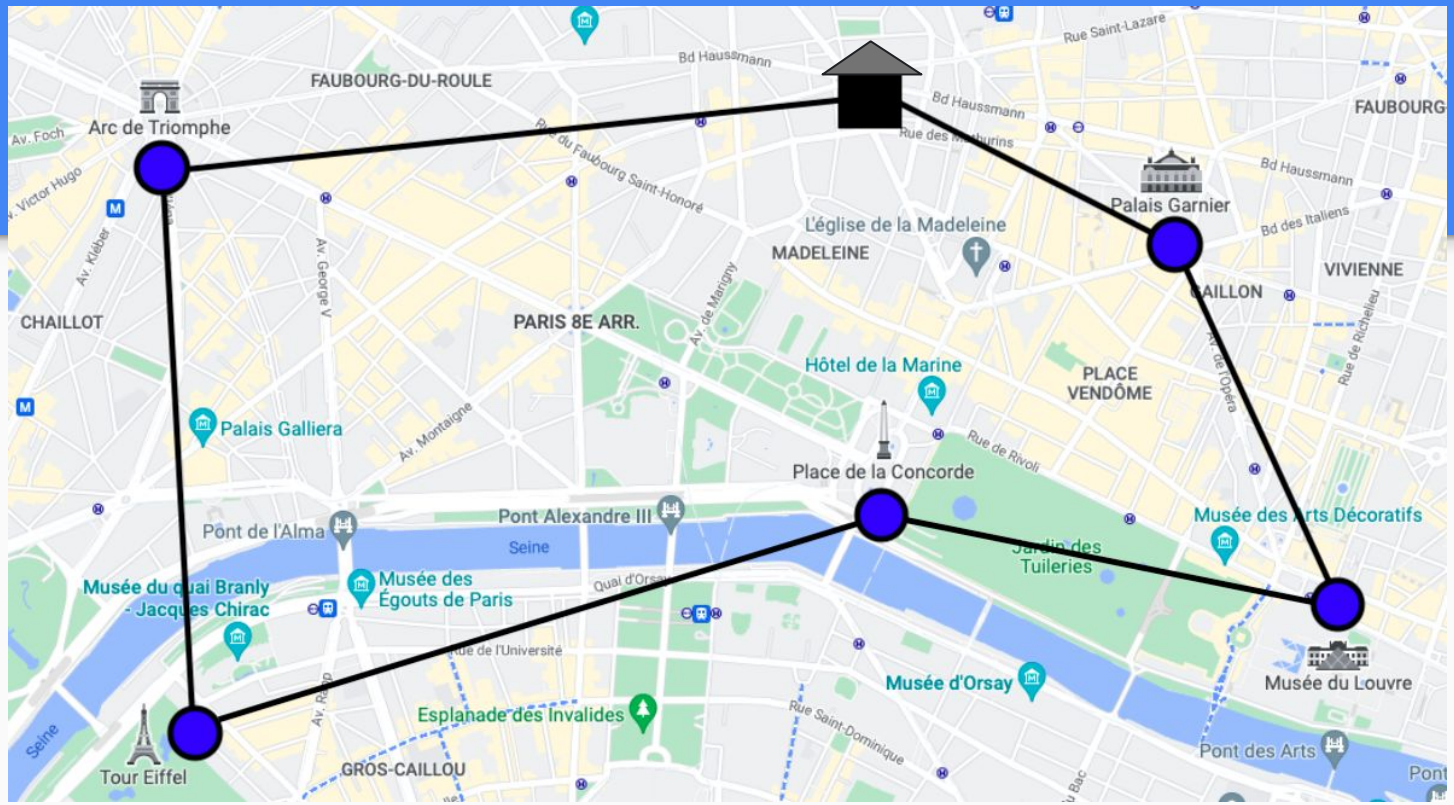
# Lien avec la santé


**Déserts médicaux**

**Ambulanciers**

https://blog.landot-avocats.net/2019/01/17/quelques-pistes-de-travail-a-travers-le-desert-medical/
https://www.europe1.fr/societe/le-transport-des-malades-perturbe-par-la-non-vaccination-des-ambulanciers-4066660

# Le problème du voyageur de commerce




- **Problème de classe NP**
- **Pour n lieux : (n-1)!/2 circuits à tester (en fixant le point de départ)**

  **Pour 11 villes : plus d'un million de possibilités**

source images : https://laurent-ott.developpez.com/tutoriels/programmation-excel-vba-tome-3/

# Comment diminuer le temps de transport d'un nombre donné de médecins tout en gérant les urgences, à l'aide d'une résolution partielle du problème du voyageur de commerce ?

**I - L'algorithme principal**

**II - Les modifications apportées pour répondre au problème posé**

**III - Comparaison de deux approches**

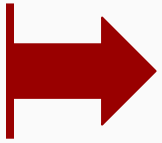
# Algorithme génétique : qu'est-ce que c'est ?

⇒ **Algorithme évolutionniste, non déterministe**

Objectifs :
→ solution **approchée**
→ **temps raisonnable**

Principe :
⇒ **sélection naturelle**

⇒ **population de solutions potentielles**
→ **sélection**
→ **croisements, mutations**



**La population va évoluer**

→ À la fin : **meilleur individu** de la population

# Les différentes classes

**Patient**
**coordonnée x**
**coordonnée y**
**priorité** (True ou False)

**ListePatient**
**liste** contenant le point de départ puis les **patients**
**nombre de patients**
**nombre de patients prioritaires**

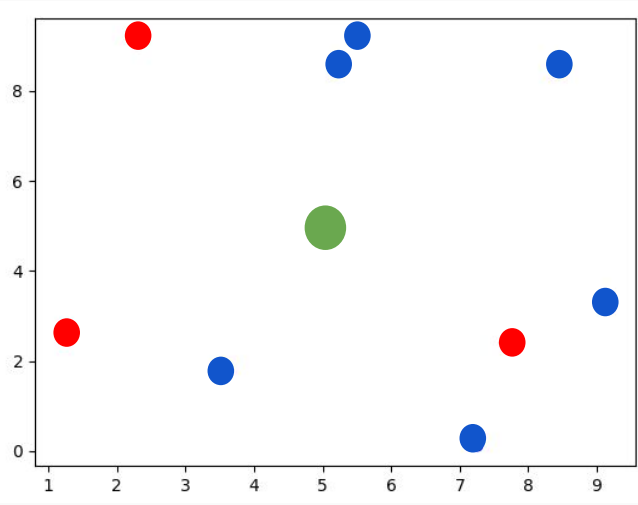
**Circuit**
**trajet** (liste de patients dans l'ordre de visite)
**distance**
**nombre de patients**

**Population**
**liste de circuits**
**taux de mutation**
**taille des poules** dans la sélection par tournoi

patient non prioritaire

patient prioritaire

point de départ




0 <= x <= 10
0 <= y <= 10

# Structure de l'algorithme

**On commence par créer une liste de patients :**

Fonctions :

**nouveauPatient**
→ entrée : priorité du patient
→ action : choisir au hasard deux coordonnées x et y entre 0 et 10
→ sortie : élément de classe **Patient**

**generateurPatient**
→ entrée : le nombre de patients et le nombre de prioritaires voulus
→ action : générer **nb_patient** patients à l'aide de **nouveauPatient**
→ sortie : élément de classe **ListePatient**

# Structure de l'algorithme

**Entrée**

Liste de patients
Démographie
Nombre de générations

**Initialisation**

Génération d'une population de circuits

Boucle for (nombre de générations)

**Renouvellement de la population**

**Élitisme**

Boucle for (démographie)

→ **Sélection par tournoi** de deux circuits
→ **Croisement** de ces circuits
→ **Mutation** possible

→ Ajout à la nouvelle population

**Sortie**

Circuit solution approchée du problème

# Structure de l'algorithme

## Sélection par tournoi

Paramètre : la taille des poules **p**

Principe de la sélection :

→ Piocher au hasard **p** circuits dans la population

→ Garder celui de plus faible distance

## Croisement

Exemple :

Parent1 = [Paris, Lyon, Marseille, Nantes, Lille]
Parent2 = [Marseille, Lille, Paris, Lyon, Nantes]

On choisit les indices de croisement :

Parent1 = [Paris, Lyon, Marseille, Nantes, Lille]

Enfant = [Paris, Marseille, Lyon, Nantes, Lille]

Parent2 = [Marseille, Lille, Paris, Lyon, Nantes]

# Structure de l'algorithme

## Mutation

Paramètre : le taux de mutation **t**

Principe :

→ Choisir un réel au hasard entre 0 et 1

→ S'il est inférieur à t, on échange deux villes dans le circuit

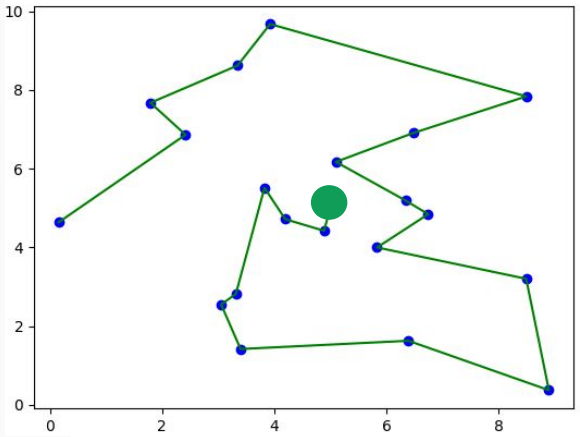
# Illustration du non déterminisme

Mêmes paramètres

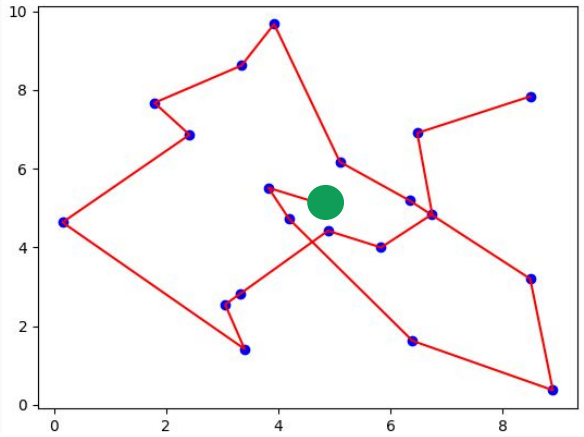
Même liste de patients

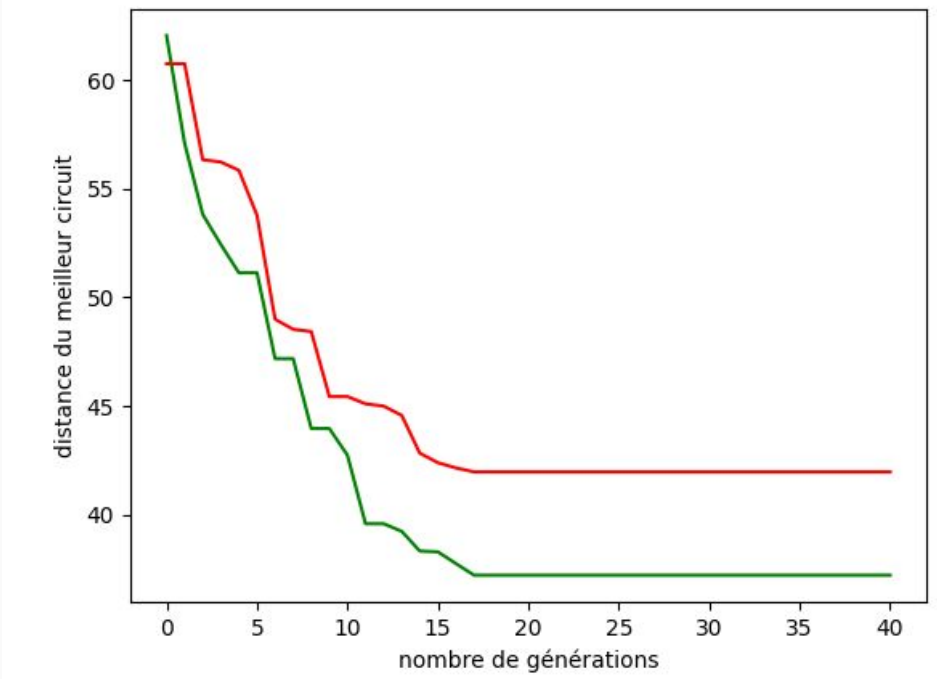
→ Circuits différents de distances différentes

**Circuit 1**



**Circuit 2**





# II - Les modifications apportées pour répondre au problème posé

**Modélisation**

**⇒ Nécessité d'une échelle de temps**

**Estimations** :
→ Consultation = 30 min
→ Vitesse moyenne = 16,2 km/h
→ Superficie à couvrir = 33 km² par médecin
→ Journée = 12h

**Module pygame**

→ Interface
→ Gérer le temps

# Gérer les nouvelles consultations

- Fonction **nouvelle_consultation** : ajoute un patient **prioritaire**

- Fonction **update**

**Entrée** : Constante de temps dt

Si le médecin est très proche du patient suivant :

→ Suppression du patient précédent dans les listes
→ Actualisation du nombre de prioritaires
→ **S'il y a une nouvelle consultation : calcul du nouveau circuit**
→ Le médecin reste pendant 30 minutes chez le patient

Si le médecin n'est pas arrêté chez un patient :

→ Déplacement du médecin

Si la journée n'est pas terminée :

→ Possiblement une nouvelle consultation

- Fonction **affichage** : affiche l'écran avec les nouvelles données

# Adaptation à n médecins

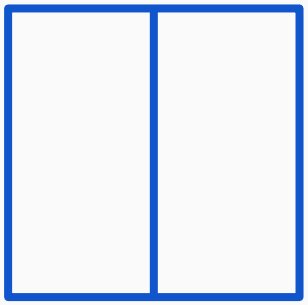
Implémentation : **liste de n médecins**

Objectif : Diviser le plan en **n** zones égales

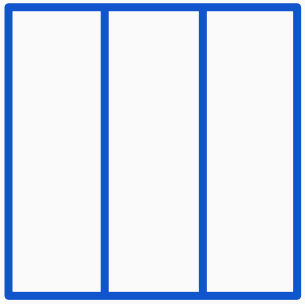
Fonction **split_patient**
→ entrée : patient, nombre de médecin (**entre 1 et 4**)
→ sortie : indice du médecin responsable de la zone

⇒ Diviser la liste de patients initiale
⇒ Adapter **nouvelle_consultation**

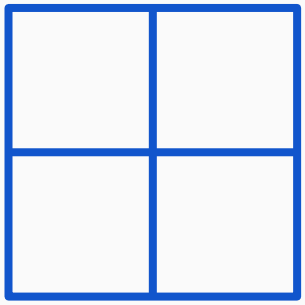
2 médecins



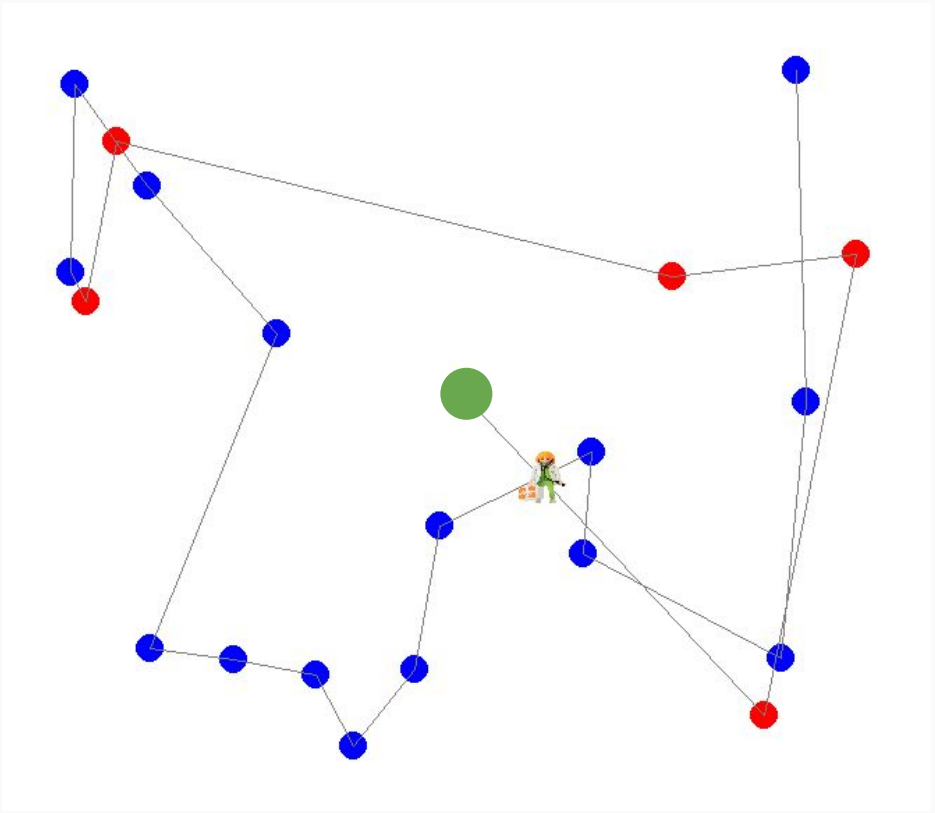
3 médecins

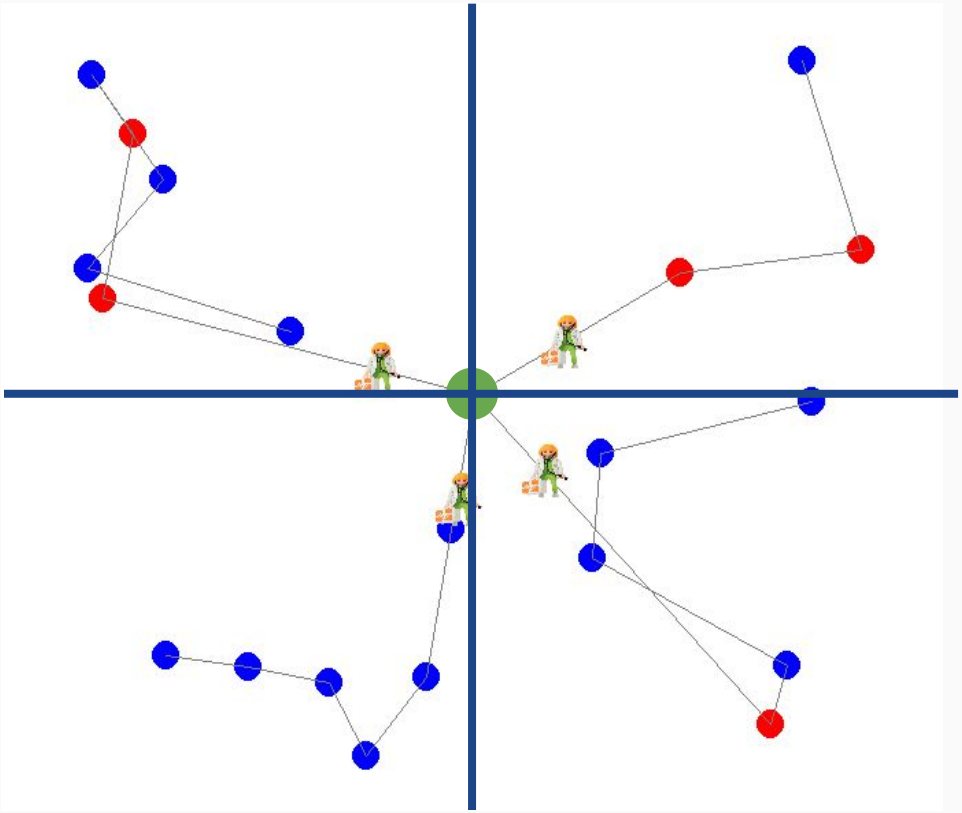


4 médecins



# Application de split_patient pour 4 médecins

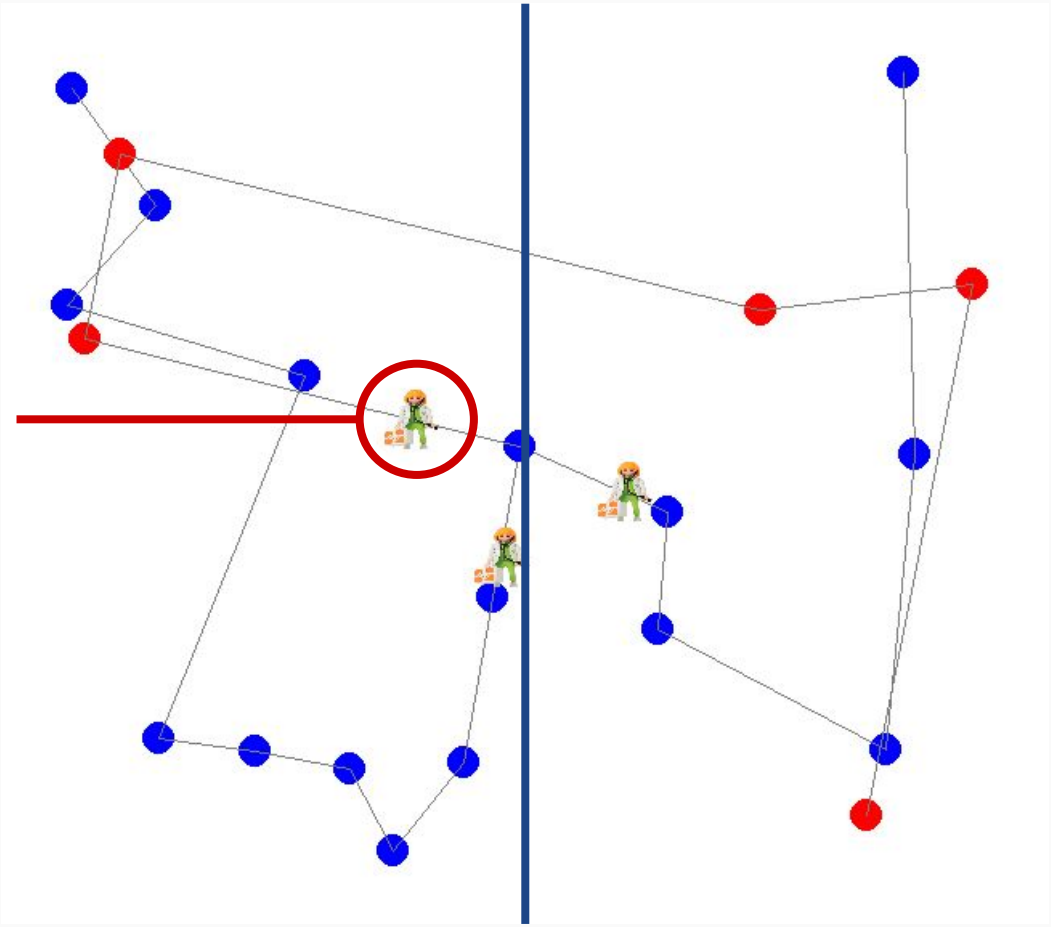




# Adaptation à un médecin qui gère les urgences

Modifier **split_patient** :
→ Associer tous les patients prioritaires à un médecin

3 médecins :



# III - Comparaison de deux approches

## pour n médecins

Première approche :
⇒ Division du plan en **n** zones égales

→ un médecin = une zone

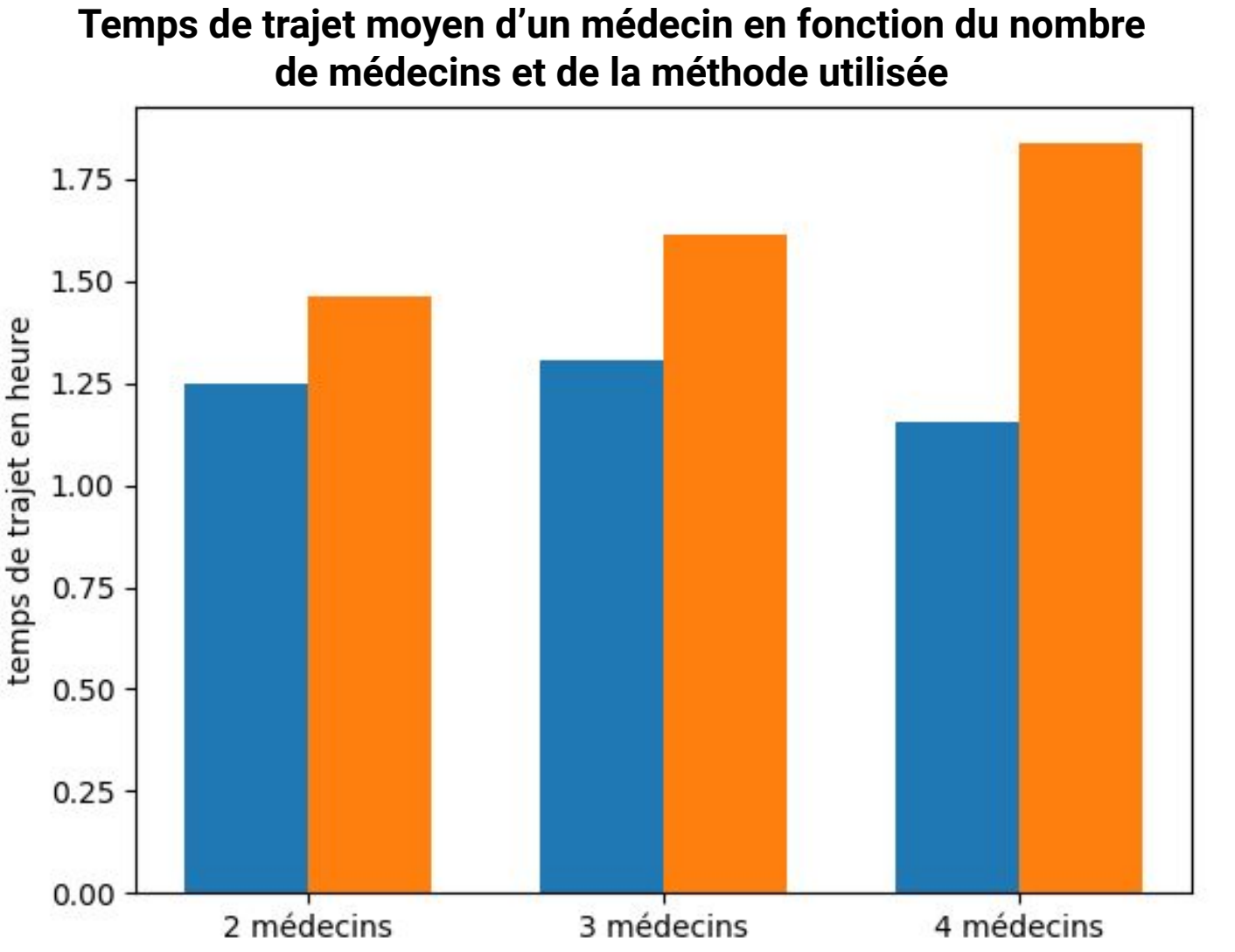
Deuxième approche :
⇒ Division du plan en **(n-1)** zones égales

→ un médecin = **urgences uniquement**
→ pour les autres : un médecin = une zone

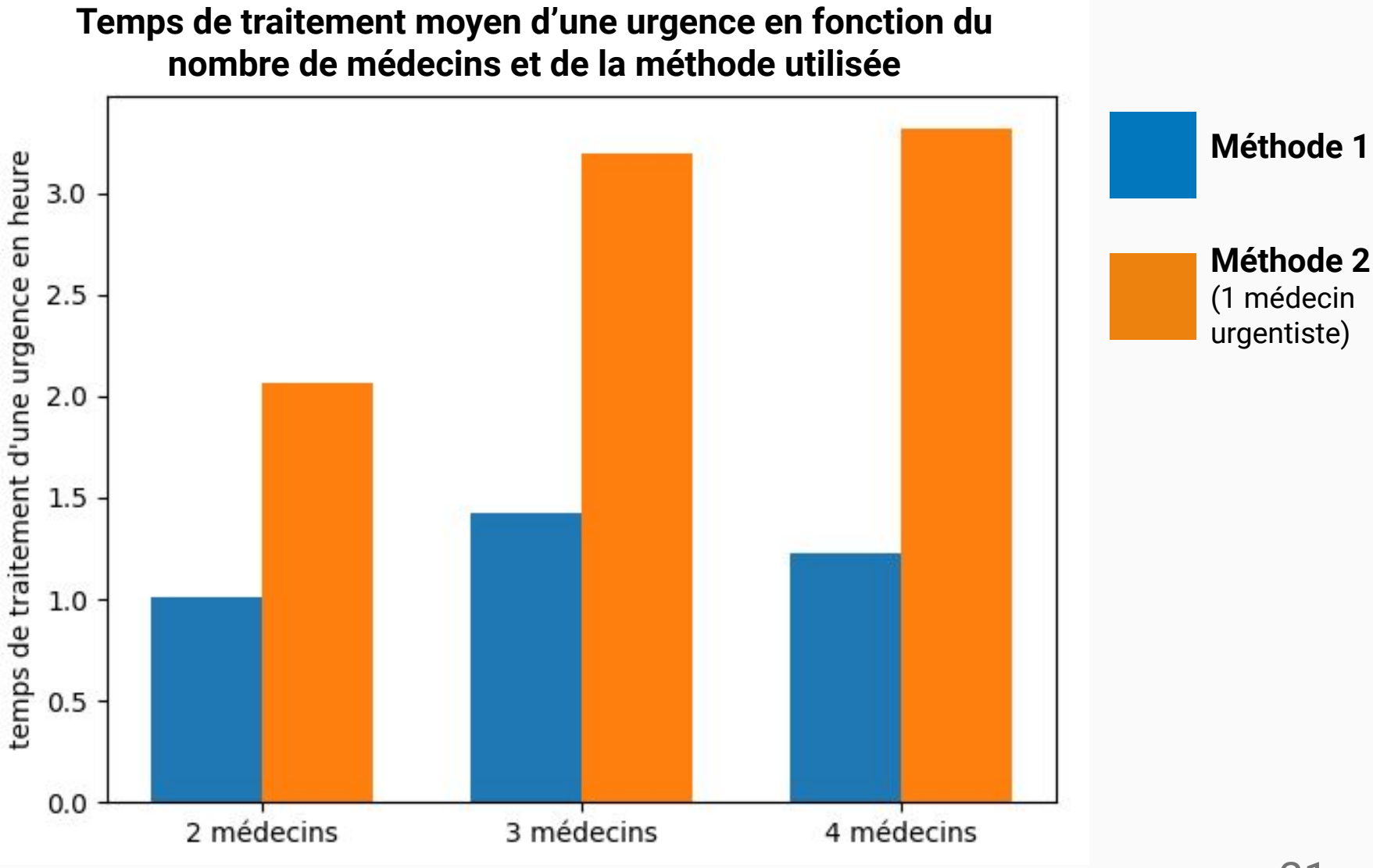
# Mesures

| Nombre de médecins | Nombre initial de patients | Superficie couverte |
|---|---|---|
| **2** | Prioritaires : **1**<br>Non prioritaires : **12** | 66 km² |
| **3** | Prioritaires : **1**<br>Non prioritaires : **18** | 99 km² |
| **4** | Prioritaires : **1**<br>Non prioritaires : **24** | 132 km² |

# Résultats


Temps de trajet moyen d'un médecin en fonction du nombre de médecins et de la méthode utilisée
temps de trajet en heure
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2 médecins
3 médecins
4 médecins
Méthode 1
Méthode 2
(1 médecin urgentiste)

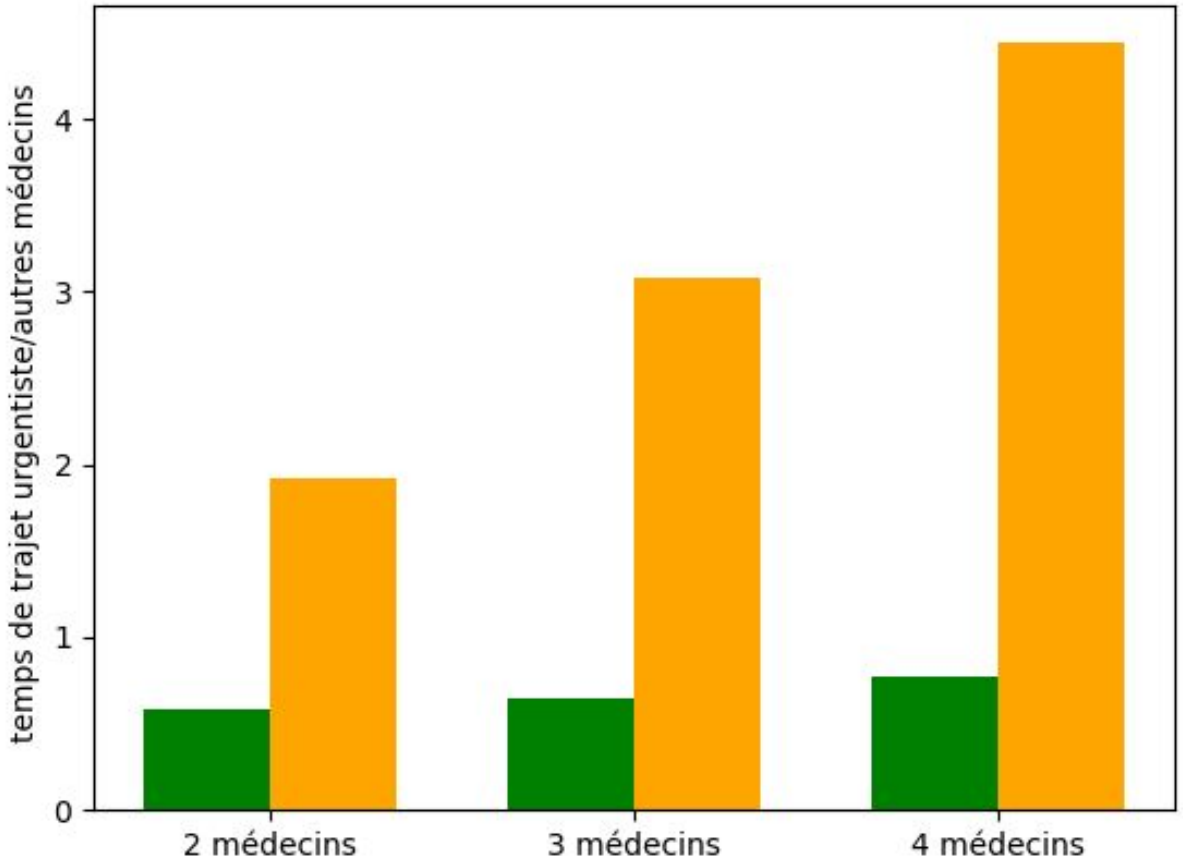
# Résultats


Temps de traitement moyen d'une urgence en fonction du nombre de médecins et de la méthode utilisée
temps de traitement d'une urgence en heure
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2 médecins
3 médecins
4 médecins
Méthode 1
Méthode 2
(1 médecin urgentiste)

# Résultats

**Méthode 2**

**Temps de trajet moyen du médecin urgentiste comparé à celui d'un autre médecin**



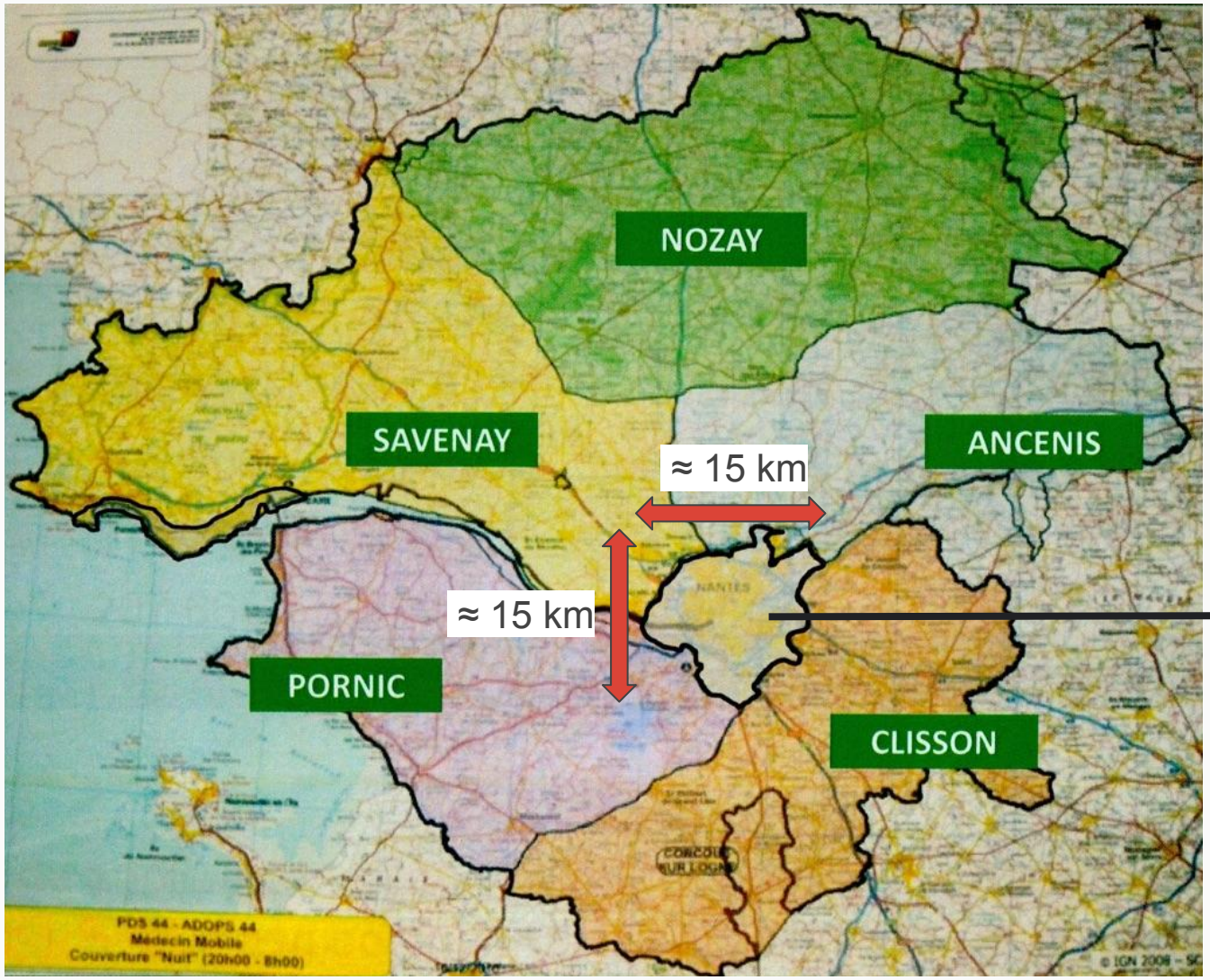
# Conclusion

- Meilleure méthode = **méthode 1** → **moins de temps de trajet**
  → **urgences traitées plus rapidement**
- Inconvénient : **impossibilité de fixer un rendez-vous** pour les patients non prioritaires

**Limite du modèle :** différents degrés d'urgence

**Difficulté rencontrée :** estimations réalistes

# Annexe 1 : Justification des estimations




Données sur SOS médecins sur le secteur de NANTES :

→ ≈ 60 interventions par nuit
→ 6 médecins

**≈ 10 interventions par médecin**

Secteur couvert par SOS médecin dans la zone de NANTES

Superficie ≈ 200 km²

Source : ADOPS 44

# Annexe 1 : Justification des estimations

Superficie couverte par :

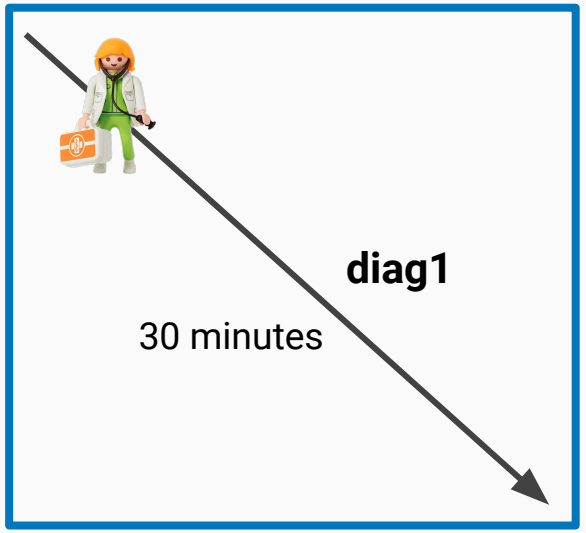
1 médecin → 33 km²

2 médecins → 66 km²

3 médecins → 99 km²

4 médecins → 132 km²

**1 médecin**




**diag1 = 8,1 km**
**v1 = 16,2 km/h**

| | diag (km) | vitesse (km/h) |
|---|---|---|
| 1 med | 8,1 | 16,2 |
| 2 med | 11,5 | 11,4 |
| 3 med | 14,1 | 9,3 |
| 4 med | 16,2 | 8,1 |

Formule :
v = diag1/(diag/v1)

**On ne change pas la superficie mais la vitesse !**

# Annexe 2 : Mesures

| Nombre de médecins | Nombre initial de patients |
|---|---|
| 2 | Prioritaires : 1<br>Non prioritaires : 12 |
| 3 | Prioritaires : 1<br>Non prioritaires : 18 |
| 4 | Prioritaires : 1<br>Non prioritaires : 24 |

→ **4 listes de patients différentes pour chaque cas**

→ 6 patients non prioritaires par médecin
→ proba_nouvelle_consultation augmente avec le nombre de médecin

```
456    proba_nouvelle_consultation = 0.0001*(nb_medecin)**0.5
```

# Annexe 3 : les modules

Importation des modules :

```
4   import random as rd
5   import matplotlib.pyplot as plt
6   import pickle
7   import pygame
```

# Annexe 4 : les classes

```
10  class Patient:
11
12      def __init__(self, x, y, priorité):
13          self.x = x
14          self.y = y  #coordonnées du point
15          self.priorité = priorité
16
17      def distance(self, patient):   #distance entre deux patients
18          return ((patient.x - self.x)**2 + (patient.y - self.y)**2)**0.5
19
20      def nouveauPatient(prioritaire):
21          '''prioritaire = true si le patient est prioritaire, false sinon'''
22          x = rd_nouvelle_consultation.uniform(0, x_max)
23          y = rd_nouvelle_consultation.uniform(0, y_max)
24          return Patient(x, y, prioritaire)
25
26      def couleur(self):
27          if self.priorité:
28              return (255, 0, 0)
29          return (0, 0, 255)
30
31      def affichagepy(self, screen):
32          pygame.draw.circle(screen, self.couleur(), (self.x *60 + 50, self.y *60 + 50), 10)
33
```

# Annexe 4 : les classes

```
36  class ListePatient:
37
38      def __init__(self, listePatient, nb_prioritaire):
39          self.liste = listePatient
40          self.nb_prioritaire = nb_prioritaire
41
42      def nb_patient(self):
43          return len(self.liste) - 1
44
45      def generateurPatient(nb_patient, nb_prioritaire):
46          point_de_depart = Patient(x_depart, y_depart, False) #par défaut
47          liste = [point_de_depart]
48          for i in range(nb_prioritaire):
49              liste.append(Patient.nouveauPatient(True))
50          for i in range(nb_patient - nb_prioritaire):
51              liste.append(Patient.nouveauPatient(False))
52          return ListePatient(liste, nb_prioritaire)
53
54      def affichage(self):
55          X = [self.liste[i].x for i in range(self.nb_patient()+1)]
56          Y = [self.liste[i].y for i in range(self.nb_patient()+1)]
57          plt.scatter(X[1:self.nb_prioritaire+1], Y[1:self.nb_prioritaire+1], c='red')
58          plt.scatter(X[self.nb_prioritaire+1:], Y[self.nb_prioritaire+1:], c='blue')
59          plt.scatter(X[0], Y[0], c='green')
60          plt.show()
```

# Annexe 4 : les classes

```
63  class Circuit:
64
65      def __init__(self, trajet, distance):
66          self.trajet = trajet.copy() #Résolution d'un bug
67          self.distance = distance
68
69      def nb_patient(self):
70          return len(self.trajet) - 1
71
72      def distance(trajet):
73          dist = 0
74          for i in range(len(trajet)-1) :
75              dist += Patient.distance(trajet[i], trajet[i+1])
76          return dist
77
78      def nouveauCircuit(listePatient):
79          trajetPatient = listePatient.liste[1:] #Le point de départ reste à sa place
80          rd.shuffle(trajetPatient)
81          trajet = [listePatient.liste[0]] + trajetPatient
82          circuit = Circuit(trajet, Circuit.distance(trajet))
83          #le premier point est le point de départ, puis on ajoute la liste des patients mélangée aléatoirement
84          return circuit
```

# Annexe 4 : les classes

```
86      def affichage(self):
87          X = [self.trajet[i].x for i in range(self.nb_patient()+1)]
88          Y = [self.trajet[i].y for i in range(self.nb_patient()+1)]
89          plt.scatter(X[1:self.nb_prioritaire+1], Y[1:self.nb_prioritaire+1], c='red')
90          plt.scatter(X[self.nb_prioritaire+1:], Y[self.nb_prioritaire+1:], c='blue')
91          plt.scatter(X[0], Y[0], c='green')
92          plt.plot(X, Y, 'grey')
93
94      def generateurCircuit(listePatient, demographie):
95          listeCircuit = []
96          for i in range(demographie) :
97              listeCircuit.append(Circuit.nouveauCircuit(listePatient))
98          return listeCircuit
99
100     def affichagepy(self, screen):
101         listePoints = [(60*patient.x + 50, 60*patient.y + 50) for patient in self.trajet]
102         pygame.draw.lines(screen, (125, 125, 125), False, listePoints)
```

# Annexe 4 : les classes

```
105 class Population:
106 
107     def __init__(self, listeCircuit):
108         self.listeCircuit = listeCircuit
109         self.tauxMutation = 0.015
110         self.tailleTournoi = 5
111 
112     def getElite(self) :
113         '''renvoie le meilleur circuit (celui de distance la plus faible)
114         de la population'''
115         meilleur_circuit = (self.listeCircuit)[0]
116 
117         for i in range(1, len(self.listeCircuit)):
118             if ((self.listeCircuit)[i]).distance < meilleur_circuit.distance :
119                 meilleur_circuit = (self.listeCircuit)[i]
120 
121         return meilleur_circuit
122 
123     def selectionTournoi(self):
124         '''renvoie le circuit avec la plus faible distance dans une poule
125         de self.tailleTournoi circuits pris au hasard dans la population'''
126         meilleur_circuit = rd.choice(self.listeCircuit)
127         for i in range(self.tailleTournoi - 1):
128             circuit = rd.choice(self.listeCircuit)
129             if circuit.distance < meilleur_circuit.distance :
130                 meilleur_circuit = circuit
131         return meilleur_circuit
```

# Annexe 4 : les classes

```
133     def croisement(self, parent1, parent2):
134         #Le point de départ est commun à tous les circuits, donc l'indice
135         #du début du croisement est supérieur à 1.
136         #Il faut que le trajet contienne au moins
137         debut = rd.randint(1, parent1.nb_patient())
138 
139         fin = rd.randint(debut+1, parent1.nb_patient()+1)
140 
141         embryonT = parent1.trajet[:debut] + parent1.trajet[fin:]
142         enfantT = parent1.trajet[:debut]
143 
144         for patient in parent2.trajet :
145             if patient not in embryonT :
146                 enfantT.append(patient)
147         enfantT += parent1.trajet[fin:]
148 
149         enfant = Circuit(enfantT, Circuit.distance(enfantT))
150         return enfant
151 
152     def mutation(self, circuit):
153         '''échange de deux points dans un circuit avec une probabilité de self.tauxMutation'''
154         if rd.random() < self.tauxMutation :
155             indice_mut = rd.randint(1, len(circuit.trajet) - 2) #On ne change pas le point de départ
156 
157             circuit.trajet[indice_mut], circuit.trajet[indice_mut+1] = circuit.trajet[indice_mut+1], circuit.trajet[indice_mut]
158             circuit.distance = Circuit.distance(circuit.trajet)
159         return circuit
```

# Annexe 5 : l'algorithme génétique

```
163  #On applique cette fonction uniquement à des listes de patients contenant des patients non prioritaires
164  #On concatène ensuite la liste de patients prioriataire avec le circuit obtenu
165  def evolutionPopulation(listePatient, demographie = 1000, nb_generation = 40) :
166
167      #S'il n'y a que le point de départ :
168      if listePatient.nb_patient() == 0 :
169          return Circuit(listePatient.liste, 0)
170
171      #S'il n'y a qu'un seul patient dans la liste :
172      if listePatient.nb_patient() == 1 :
173          return Circuit(listePatient.liste, Circuit.distance(listePatient.liste))
174
175
176      listeCircuit = Circuit.generateurCircuit(listePatient, demographie)
177      population = Population(listeCircuit)
178
179      meilleurCircuit = Population.getElite(population)
180      DistanceMeilleurCircuit = [meilleurCircuit.distance]
```

# Annexe 5 : l'algorithme génétique

```
182     for i in range(nb_generation):
183         nouveauxCircuits = [meilleurCircuit]
184 
185         for k in range(demographie - 1):
186             parent1 = Population.selectionTournoi(population)
187             parent2 = Population.selectionTournoi(population)
188 
189             enfant = Population.croisement(population, parent1, parent2)
190 
191             enfant = Population.mutation(population, enfant)
192 
193             nouveauxCircuits.append(enfant)
194 
195         population = Population(nouveauxCircuits)
196 
197         meilleurCircuit = Population.getElite(population)
198         DistanceMeilleurCircuit.append(meilleurCircuit.distance)
199 
200     return meilleurCircuit
```

# Annexe 6 : algorithme général

Fonction d'affichage :

```
205  def affichage(screen) :
206      global nb_medecin
207      for i in range(nb_medecin):
208          for patient in P[i].liste :
209              patient.affichagepy(screen)
210          if len(C[i].trajet) > 1 :
211              C[i].affichagepy(screen)
212
213          screen.blit(medecinIMG, (60 * Med[i].x + 50 - 20, 60 * Med[i].y + 50 - 20))
```

# Annexe 6 : algorithme général

Fonction qui fait apparaître des nouvelles consultations :

```
217  def nouvelle_consultation(dt):
218      #Les nouvelles consultations sont forcément des urgences
219
220      if rd_nouvelle_consultation.random() < dt*proba_nouvelle_consultation :
221          patient = Patient.nouveauPatient(True)
222          indice_med = split_patient(patient, nb_medecin - urgence, urgence)
223
224          #Si une urgence apparait avant le départ du médecin du point de départ,
225          #on insère l'urgence après le point de départ (résolution d'un bug)
226          if P[indice_med].liste[0].x == 5 and P[indice_med].liste[0].y == 5 :
227              P[indice_med].liste.insert(1, patient)
228
229          else :
230              P[indice_med].liste.insert(P[indice_med].nb_prioritaire +1, patient)
231
232          P[indice_med].nb_prioritaire += 1
233
234          #Si le médecin est urgentiste, ou si tous les patients sont prioritaires,
235          #on ajoute le patient à la suite des autres urgences
236          if urgence == 1 or P[indice_med].nb_patient() == P[indice_med].nb_prioritaire :
237              C[indice_med].trajet.append(patient)
238
239          #Sinon, on calcule de nouveau le circuit
240          else :
241              indic_calcul_circuit[indice_med] = True
242
243          indic_arret[indice_med] = False
244
245          temps_urgence[patient] = 0
```

# Annexe 6 : algorithme général

Fonction update, qui actualise la position des médecins, leur temps de trajet et le temps de traitement des urgences :

```
250  def update(dt) :
251      global journee
252
253      for i in range(nb_medecin) :
254          if indic_arret[i] :
255              continue
256
257          if len(C[i].trajet) == 1 and indic_calcul_circuit[i] :
258              C[i] = Circuit(P[i].liste, Circuit.distance(P[i].liste))
259
260          if Med[i].distance(C[i].trajet[1])<0.1 :
261              if len(P[i].liste) == 2 :
262                  indic_arret[i] = True
263                  continue
264              patient_precedent = C[i].trajet.pop(0)  #On enlève le patient duquel le médecin est parti
265              #P[i].liste.remove(patient_precedent)
266              for j in range(len(P[i].liste)):
267                  patient = P[i].liste[j]
268                  if patient == patient_precedent:
269                      P[i].liste.pop(j)
270                      break
271              if C[i].trajet[0].priorité :
272                  P[i].nb_prioritaire -= 1
273                  #On actualise le nombre de prioritaires
274                  #(si le patient chez lequel le médecin arrive devient le point de départ,
275                  #il n'est donc plus prioritaire)
276
```

# Annexe 6 : algorithme général

```
277             P[i].liste.remove(C[i].trajet[0])
278             P[i].liste.insert(0, C[i].trajet[0])
279             #On place le patient chez lequel on arrive comme point de départ
280
281             if indic_calcul_circuit[i] :
282                 #Indice qui sépare les patients prioritaires des non prioritaires
283                 #Le point de départ du trajet des patients non prioritaires devient le dernier patient urgent
284                 indice_separation = P[i].nb_prioritaire
285                 circuit = evolutionPopulation(ListePatient(P[i].liste[indice_separation:], 0))
286                 trajet = P[i].liste[:indice_separation] + circuit.trajet
287                 C[i] = Circuit(trajet, Circuit.distance(trajet))
288                 indic_calcul_circuit[i] = False
289
290             timer[i] = temps_consultation
```

# Annexe 6 : algorithme général

```
294     for i in range(nb_medecin):
295         if timer[i] > 0 :
296             timer[i] -= dt
297         elif not indic_arret[i] :
298             tempsMed[i] += dt
299 
300             norme_vect_dir = Med[i].distance(C[i].trajet[1])
301             vect_normalise = ((C[i].trajet[1].x - Med[i].x)/norme_vect_dir, (C[i].trajet[1].y - Med[i].y)/norme_vect_dir)
302             vect_vitesse = (vect_normalise[0]*vitesse_pixel_dt, vect_normalise[1]*vitesse_pixel_dt)  #"fois un dt"
303 
304             #On retranscrit les coordonnées dans une zone de 10 par 10
305             Med[i].x += vect_vitesse[0]/60
306             Med[i].y += vect_vitesse[1]/60
307 
308             if urgence == 0 :
309                 for patient in P[i].liste :
310                     if patient.priorité :
311                         temps_urgence[patient] += dt
312 
313     if urgence == 1 and not indic_arret[nb_medecin - 1] :
314         for patient in P[nb_medecin-1].liste[1:] :
315             temps_urgence[patient] += dt
316         #On ajoute dt au temps de traitement de tous les patients non encore vus
317 
318 
319     journee -= dt
320     if journee <= 0 :
321         return
322 
323     nouvelle_consultation(dt)
```

# Annexe 6 : algorithme général

Fonction qui associe au patient pris en entrée l'indice du médecin responsable de lui :

```
352  def split_patient(patient, nb_medecin, urgence) :
353      if urgence == 1 :
354          if patient.priorité :
355              return nb_medecin
356              #On met le médecin assigné aux urgences après les autres médecins dans la liste de médecins
357      indice_med = 0
358      if nb_medecin % 2 == 0 :
359          if patient.x < 5 :
360              indice_med = 0
361          else :
362              indice_med = 1
363          #Si le nombre de médecins est divisible par 2, on divise d'abord la zone en 2
364
365      if nb_medecin == 4 :
366          if patient.y < 5 :
367              indice_med = 2*indice_med
368          else :
369              indice_med = 2*indice_med + 1
370          #Si le nombre de médecins vaut 4, on divise de nouveau les zones en 2
371
372      if nb_medecin == 3 :
373          if patient.x < 3.33 :
374              indice_med = 0
375          elif patient.x < 6.66 :
376              indice_med = 1
377          else :
378              indice_med = 2
379
380      return indice_med
```

# Annexe 6 : algorithme général

Fonction  qui réparti les patients initiaux entre chaque médecin, en divisant la liste de patients initiale en plus petites listes :

```
328  def split_listePatient(Pinit):
329      global nb_medecin
330      global urgence
331      #Ne fonctionne que pour nb_medecin = 2, 3, 4
332      liste = [[Pinit.liste[0]] for i in range(nb_medecin)]
333      #Le point de départ est commun à tous les médecins
334      nb_prioritaire = [0 for i in range(nb_medecin)]
335      #Initialisation
336
337      for patient in Pinit.liste[1:] :
338          indice_med = split_patient(patient, nb_medecin - urgence, urgence)
339          if patient.priorité :
340              liste[indice_med].insert(1, patient)
341              nb_prioritaire[indice_med] += 1
342          else :
343              liste[indice_med].append(patient)
344
345      P = []
346      for i in range(nb_medecin):
347          P.append(ListePatient(liste[i], nb_prioritaire[i]))
348
349      return P
```

# Annexe 6 : algorithme général

Initialisation du plan :

```
##Initialisation

x_max = 10 #le secteur couvert
y_max = 10

x_depart = 5 #point de départ du médecin
y_depart = 5
```

Importation de l'image du médecin, et des listes de patients initiales qui vont servir pour les mesures :

```
#On charge leur image
medecinIMG = pygame.image.load('medecin.png')
medecinIMG = pygame.transform.scale(medecinIMG, (40,40))

#Listes de patients initiales utilisées
MED2 = [pickle.load(open('2med_l1', 'rb')), pickle.load(open('2med_l2', 'rb')), pickle.load(open('2med_l3', 'rb')), pickle.load(open('2med_l4', 'rb'))]

MED3 = [pickle.load(open('3med_l1', 'rb')), pickle.load(open('3med_l2', 'rb')), pickle.load(open('3med_l3', 'rb')), pickle.load(open('3med_l4', 'rb'))]

MED4 = [pickle.load(open('4med_l1', 'rb')), pickle.load(open('4med_l2', 'rb')), pickle.load(open('4med_l3', 'rb')), pickle.load(open('4med_l4', 'rb'))]

MED = [MED2, MED3, MED4]
```

# Annexe 6 : algorithme général

```
410  ##Données
411
412  journee = 72000 #12h
413
414  temps_consultation = 3000 #30 min
415
416  dt = 10
```

Fonction qui prend en entrée le dictionnaire temps_urgence (*dont les clés sont les patients prioritaires et dont les valeurs sont le temps de traitement de ces patients*) et qui renvoie le temps minimal, maximal et moyen de traitement d'une urgence :

```
420  ##Mesures
421
422  def temps_urgence_min_max_moy(temps_urgence):
423      min = 144000 #durée d'une journée multipliée par 2
424      max = 0
425      somme = 0
426      for temps in temps_urgence.values() :
427          if temps < min :
428              min = temps
429          if temps > max :
430              max = temps
431          somme += temps
432      moy = somme/len(temps_urgence.values())
433      return min, max, moy
434
```

# Annexe 6 : algorithme général

```
438  #Variables
439  nb_medecin = 4
```

Constantes :

```
440  nb_test = 4
441
442  #300dt = 30 min
443  #1 heure = 600dt
444  #coef heure/dt = 600
445  coef_dt_h = 1/600
446
447  #équivalence pixel/mètre :
448  #600 pixels = 5,74 km
449  #coef pixel/km = 600/5,74
450  coef_p_km = 600/5.74
451
452  #Au lieu de changer la superficie en fonction du nombre de médecin, on fait varier la vitesse
453  vitesse = [16.2, 11.4, 9.3, 8.1]
454
455  #On convertit la vitesse en km/h en pixel/dt
456  vitesse_pixel_dt = vitesse[nb_medecin-2]*coef_dt_h*coef_p_km
```

# Annexe 6 : algorithme général

Plus il y a de médecins, plus il y a de nouvelles consultations qui apparaissent :

```
459   proba_nouvelle_consultation = 0.0001*(nb_medecin)**0.5
460
```

Initialisation de l'aléatoire pour les nouvelles consultations :

```
462   rd_nouvelle_consultation = rd.Random(1)
```

Initialisation des listes qui vont contenir les temps de trajet et les temps de traitement des urgences :

```
465   TM = []
466   TU_min_max_moy = []
```

# Annexe 7 : algorithme de prise des mesures

```
468 for j in range(2) :
469     #2 méthodes
470 
471     urgence = j
472 
473     TM0 = []
474     #temps de trajet de chaque médecin avec la méthode j
475     TU_min_max_moy0 = [[], [], []]
476     #temps de traitement de chaque urgence avec avec la méthode j
477 
478     for i in range(nb_test) :
479 
480         #On définit une durée maximale de la journée, ici on a choisi 10h
481         journee = 60000
482 
483         #On replace les médecins au point de départ
484         Med = [Patient(5, 5, False) for k in range(nb_medecin)]
485 
486         #Permet de ne pas recalculer le circuit à chaque itération,
487         # mais seulement lorsqu'une nouvelle consultation est ajoutée
488         indic_calcul_circuit = [False for k in range(nb_medecin)]
489 
490         #Lorsque le médecin arrive près d'un patient, on associe à timer une valeur
491         # (ici on a choisi 2000ms ce qui correspond à 20 minutes)
492         timer = [0 for k in range(nb_medecin)]
493 
494         #Utile lorsqu'il il n'y a plus de patient à voir
495         indic_arret = [False for k in range(nb_medecin)]
```

# Annexe 7 : algorithme de prise des mesures

```
497         #Pour mesurer le temps de trajet de chaque médecin
498         tempsMed = [0 for k in range(nb_medecin)]
499
500
501
502         #On définit la liste de patient initiale
503         Pinitial = MED[nb_medecin - 2][i]
504
505         #On définit la liste de patient initiale par médecin
506         P = split_listePatient(Pinitial)
507
508         #On calcule le circuit initial pour chaque médecin
509         #Il y a toujours une seule urgence initialement
510         C = []
511
512         if urgence == 0 :
513             for k in range(nb_medecin):
514                 if P[k].nb_prioritaire == 1 :
515                     trajet = [P[k].liste[0]] + evolutionPopulation(ListePatient(P[k].liste[1:], 1)).trajet
516                     C.append(Circuit(trajet, Circuit.distance(trajet)))
517                 else :
518                     C.append(evolutionPopulation(P[k]))
519
520         else :
521             for k in range(nb_medecin-1):
522                 C.append(evolutionPopulation(P[k]))
523             C.append(Circuit(P[nb_medecin-1].liste, Circuit.distance(P[nb_medecin-1].liste)))
524
```

# Annexe 7 : algorithme de prise des mesures

```
526         #Pour calculer le temps moyen de traitement d'une urgence,
527         # on calcule le temps de traitement de chaque urgence.
528         #Pour cela, on crée un dictionnaire.
529         #Les nouvelles urgences sont ajoutées à la liste et initialisées à 0.
530         temps_urgence = {}
531
532         if urgence == 1 :
533             for patient in P[nb_medecin-1].liste[1:] :
534                 temps_urgence[patient] = 0
535
536         elif urgence == 0 :
537             for k in range(nb_medecin) :
538                 for patient in P[k].liste[1:] :
539                     if patient.priorité :
540                         temps_urgence[patient] = 0
```

# Annexe 7 : algorithme de prise des mesures

```
543         #On souhaite que les consultations qui apparaissent soient les mêmes
544         # lors de la comparaison des deux méthodes pour une même liste de patients.
545         #Pour cela, on "fixe l'aléatoire"
546         rd_nouvelle_consultation.seed(10*nb_medecin + i)
547 
548         pygame.init()
549 
550         screen = pygame.display.set_mode([700, 700])
551 
552         running = True
553         indic_run = True
554 
555         while (journee >= 0 or indic_run) and running :
556             for event in pygame.event.get() :
557                 if event.type == pygame.QUIT :
558                     running = False
559 
560             screen.fill((255, 255, 255))
561 
562             update(dt)
563             affichage(screen)
564 
565             pygame.display.flip()
566 
567             indic_run = (sum(indic_arret) < nb_medecin)
568             #Si tous les médecins sont à l'arrêt, indic_run = False
569             #Sinon, indic_run = True
570 
571         pygame.quit()
```

# Annexe 7 : algorithme de prise des mesures

```
573             TM0.append(tempsMed)
574
575             min, max, moy = temps_urgence_min_max_moy(temps_urgence)
576             TU_min_max_moy0[0].append(min)
577             TU_min_max_moy0[1].append(max)
578             TU_min_max_moy0[2].append(moy)
579
580             temps_urgence = {}
581             #On remet le dictionnaire à 0
582
583         TM.append(TM0)
584         TU_min_max_moy.append(TU_min_max_moy0)
585
586     print(TM, TU_min_max_moy)
587
588     #On obtient deux listes :
589     ###TM[0] contient le temps de trajet de chaque médecin pour tous les tests effectués avec la méthode 1
590     #  TM[1] de même avec la méthode 2
591     ###TU_min_max_moy[0][0] contient le temps minimum de traitement d'une urgence avec la première méthode
592     #  TU_min_max_moy[0][1] donne le maximum, etc
593     #  TU[1][0, 1 ou 2] donne les mêmes informations pour la seconde méthode
```

# Annexe 8 : code pour l'illustration du non déterminisme

Modification de la fonction affichage de la classe Circuit :

```
def affichage(self, color):
    X = [self.trajet[i].x for i in range(self.nb_patient+1)]
    Y = [self.trajet[i].y for i in range(self.nb_patient+1)]
    plt.scatter(X[1:self.nb_prioritaire+1], Y[1:self.nb_prioritaire+1], c='red')
    plt.scatter(X[self.nb_prioritaire+1:], Y[self.nb_prioritaire+1:], c='blue')
    plt.scatter(X[0], Y[0], c='green')
    plt.plot(X, Y, color)
```

```
def afficher() :
    MC1, D1 = evolutionPopulation(P)
    MC2, D2 = evolutionPopulation(P)

    plt.figure(1)
    #On trace la distance du meilleur circuit en fonction de la génération
    X = [i for i in range(40 + 1)]
    plt.plot(X, D1, 'green')
    plt.plot(X, D2, 'red')
    plt.xlabel('nombre de générations')
    plt.ylabel('distance du meilleur circuit')

    plt.figure(2)
    #On trace le meilleur circuit obtenu à l'issue de toutes les générations
    Circuit.affichage(MC1, 'green')

    plt.figure(3)
    Circuit.affichage(MC2, 'red')

    plt.show()
```

# Annexe 9 : code de construction des  histogrammes

Importation des modules :

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

# Annexe 9 : code de construction des  histogrammes

Mesures effectuées :

```
54  TT2MED = [[[4300, 9320], [7680, 4420], [8430, 7740], [8390, 9620]],
    [[6000, 8570], [6350, 6280], [6100, 16670], [5780, 14490]]]
55
56  TU2MED = [[[780, 220, 890, 740], [4180, 2680, 2970, 3530],
    [2182.8571428571427, 1433.75, 1833.3333333333333,
    2001.4285714285713]], [[1070, 440, 750, 1260], [7300, 9840, 18040,
    12460], [4197.142857142857, 4581.25, 8220.666666666666,
    6886.428571428572]]]
57
58
59  TT3MED = [[[5250, 8070, 8600], [8840, 8310, 8470], [5560, 5500,
    10090], [10310, 7740, 7380]], [[4150, 5230, 15690], [5500, 5990,
    20430], [5670, 4690, 21310], [5820, 5300, 16450]]]
60
61  TU3MED = [[[10, 580, 160, 650], [3720, 3460, 3520, 4370],
    [1640.7692307692307, 2010.0, 1758.1818181818182, 2060.0]], [[960,
    1390, 480, 1510], [17980, 12100, 17280, 13520], [7631.538461538462,
    6961.428571428572, 8971.818181818182, 7911.818181818182]]]
62
63
64  TT4MED = [[[3100, 7590, 8750, 4990], [10880, 5340, 5330, 7930], [9820,
    8980, 5070, 5440], [6750, 7730, 4520, 8540]], [[5740, 4640, 5930,
    24200], [6090, 5390, 7100, 31530], [7770, 3600, 6700, 31640], [4320,
    6350, 6220, 19190]]]
65
66  TU4MED = [[[1110, 300, 530, 850], [2990, 3820, 3460, 2730],
    [1812.2222222222222, 1987.857142857143, 1781.3333333333333,
    1800.909090909091]], [[820, 880, 630, 1990], [15790, 34610, 29650,
    14870], [8075.555555555556, 17535.0, 16548.666666666668,
    7810.909090909091]]]
```

# Annexe 9 : code de construction des  histogrammes

Fonctions de calcul de moyennes, pour les histogrammes :

```
70  def temps_med_moy(TT):
71      '''prend en entrée la liste qui contient les temps de trajets des
    médecins pour chaque méthode, et renvoie le temps de trajet moyen d'un
    médecin pour la première et pour la seconde méthode'''
72      nb_medecin = len(TT[0][0])
73      nb_test = len(TT[0])
74      TMOY = []
75      for j in range(2):
76          temps_tot = 0
77          for i in range(nb_test):
78              for k in range(nb_medecin):
79                  temps_tot += TT[j][i][k]
80          TMOY.append(temps_tot/(nb_medecin*nb_test)*(1/6000))
81          #Pour mettre en heure (une heure = 6000)
82      return TMOY
83
```

# Annexe 9 : code de construction des  histogrammes

```
86  def temps_urgence(TU):
87      '''prend en entrée la liste qui contient les temps minimaux,
    maximaux, moyens de traitement des urgences pour chaque méthode, et
    renvoie le temps moyen mimal, maximal et moyen de traitement d'une
    urgence pour chaque méthode'''
88      nb_test = len(TU[0][0])
89      min_tot = 0
90      max_tot = 0
91      moy_tot = 0
92      TUMOY = []
93      for j in range(2):
94          for i in range(nb_test):
95              min_tot += TU[j][0][i]
96              max_tot += TU[j][1][i]
97              moy_tot += TU[j][2][i]
98          TUMOY.append((min_tot/nb_test*(1/6000), max_tot/
    nb_test*(1/6000), moy_tot/nb_test*(1/6000)))
99      return TUMOY
```

# Annexe 9 : code de construction des  histogrammes

```
101 def temps_urgentiste(TT):
102     '''prend en entrée la liste qui contient les temps de trajet des
    médecins pour chaque méthode, renvoie le temps de trajet moyen du
    médecin urgentiste et celui des autres médecins additionné, dans le
    cas de la seconde méthode'''
103     nb_test = len(TT[0])
104     nb_medecin = len(TT[0][0])
105     temps_urgentiste = 0
106     temps_med = 0
107     for i in range(nb_test):
108         temps_urgentiste += TT[1][i][nb_medecin-1]
109         for k in range(nb_medecin-1):
110             temps_med += TT[1][i][k]
111     return [temps_urgentiste/nb_test*(1/6000), temps_med/
    (nb_test*nb_medecin-1)*(1/6000)]
```

# Annexe 9 : code de construction des  histogrammes

Initialisation de l'histogramme :

```
128 #Histogramme tracé en fonction du nombre de médecins
129 nb_med = ['2 médecins', '3 médecins', '4 médecins']
130
131
132 # Position sur l'axe des x pour chaque étiquette
133 position = np.arange(len(nb_med))
134 # Largeur des barres
135 largeur = .35
136 # Création de la figure et d'un set de sous-graphiques
137 fig, ax = plt.subplots()
```

# Annexe 9 : code de construction des  histogrammes

Création de chaque histogramme :

```
140 ## Temps de trajet moyen
141
142 TMOY2 = temps_med_moy(TT2MED)
143 TMOY3 = temps_med_moy(TT3MED)
144 TMOY4 = temps_med_moy(TT4MED)
145
146 r1 = ax.bar(position - largeur/2, [TMOY2[0], TMOY3[0], TMOY4[0]],
    largeur)
147 r2 = ax.bar(position + largeur/2, [TMOY2[1], TMOY3[1], TMOY4[1]],
    largeur)
```

```
150 ## Temps moyen de traitement d'une urgence
151
152 TU2 = temps_urgence(TT2MED)
153 TU3 = temps_urgence(TT3MED)
154 TU4 = temps_urgence(TT4MED)
155
156 r1 = ax.bar(position - largeur/2, [TU2[0][1], TU3[0][1], TU4[0][1]],
    largeur)
157 r2 = ax.bar(position + largeur/2, [TU2[1][1], TU3[1][1], TU4[1][1]],
    largeur)
```

# Annexe 9 : code de construction des  histogrammes

```
160 ## Temps de trajet moyen du médecin urgentiste par rapport à celui des
    autres médecins, dans le cas de la seconde méthode
161
162 TTU2 = temps_urgentiste(TT2MED)
163 TTU3 = temps_urgentiste(TT3MED)
164 TTU4 = temps_urgentiste(TT4MED)
165
166 r1 = ax.bar(position - largeur/2, [TTU2[1], TTU3[1], TTU4[1]],
    largeur, color = 'green')
167 r2 = ax.bar(position + largeur/2, [TTU2[0], TTU3[0], TTU4[0]],
    largeur, color = 'orange')
168
```

Affichage des légendes :

```
170 # Modification des marques sur l'axe des x et de leurs étiquettes
171 ax.set_xticks(position)
172 ax.set_xticklabels(nb_med)
173
174 plt.ylabel('temps de trajet urgentiste/autres médecins ')
175
```