Filière MP - ENS de Paris-Saclay, Lyon, Rennes et Paris - Session 2018
Page de garde du rapport de TIPE

NOM : SCHNEIDER	Prénoms : Guillaume Vincent Jean
Classe : MP	
Lycée : Aux Lazaristes	Numéro de candidat : 2067
Ville : Lyon	

Concours auxquels vous êtes admissible, dans la banque MP inter-ENS (les indiquer par une croix) :

ENS Cachan	MP - Option MP	X	MP - Option MPI	
	Informatique			
ENS Lyon	MP - Option MP		MP - Option MPI	
	Informatique - Option M	X	Informatique - Option P	
ENS Rennes	MP - Option MP	X	MP - Option MPI	
	Informatique			
ENS Paris	MP - Option MP	X	MP - Option MPI	
	Informatique			

Matière dominante du TIPE (la sélectionner d'une croix inscrite dans la case correspondante) :

Informatique		Mathématiques	X	Physique	

Titre du TIPE : Programmation informatique d'un modèle de mouvement de foule

Nombre de pages (à indiquer dans les cases ci-dessous) :

Texte	5	Illustration	9	Bibliographie	1

Résumé ou descriptif succinct du TIPE (6 lignes, maximum) :
Le but de mon travail est de développer et implémenter informatiquement un modèle de mouvements de foule. Pour celui-ci, j'ai besoin de déterminer la vitesse souhaitée d'un individu, ce que je fait à l'aide de la Fast Marching Method; puis celle empêchant les collisions, obtenue avec la méthode de Uzawa

À Lyon

Le 14/06/2018

Signature du (de la) candidat(e)

Signature du professeur responsable de la classe préparatoire dans la discipline

Cachet de l'établissement

Centre Scolaire
"AUX LAZARISTES"
C. P. G. E.
24, Montée St Barthélémy
69321 LYON CEDEX 05
Tél. [illegible]

La signature du professeur responsable et le tampon de l'établissement ne sont pas indispensables pour les candidats libres (hors CPGE).

Programmation informatique d'un modèle de mouvements de foule

Guillaume Schneider

14 Juin 2018

Des incidents comme ceux de La Mecque en 2015 où la rencontre de deux foules a provoqué plus de 700 décès montrent l'importance de comprendre et modéliser les mouvements de foule. Ainsi, j'ai voulu développer et implémenter un modèle facilement utilisable et ne reposant sur aucune hypothèse complexe, et j'ai choisi celui proposé par [2].

Dans celui-ci, les piétons sont modélisés par des cercles rigides ayant une vitesse souhaitée dépendant uniquement de leur position et leur permettant de rejoindre le plus rapidement possible une issue, ce qui correspond à une situation de panique.

Le modèle utilise la méthode d'Euler en déterminant à chaque itération les vitesses souhaitées et l'ensemble des vitesses admissibles, qui correspond aux vitesses empêchant la superposition de deux personnes. Les vitesses effectives seront alors calculées en projetant le vecteur des vitesses souhaitées sur cet ensemble.

Table des matières

1 Détermination de la vitesse souhaitée avec la Fast Marching Method

1.1 Description

L'algorithme de FMM décrit dans [3], permet de modéliser la propagation d'une onde dans un milieu. Ainsi, il peut être utilisé pour construire une carte des distances à la sortie. La distance d'un point à cette dernière correspond au temps de propagation d'une onde de la sortie à ce point.

Cet algorithme utilise le même principe que l'algorithme de Dijkstra : on calcule de proche en proche les distances de chaque point à l'origine en acceptant comme calculé à chaque itération le point dont la distance est la plus faible, puis en déterminant la distance à l'origine de ses voisins.

La structure en pseudo-code de l'algorithme est la suivante :

```
for all points en sortie do
  déterminer leur distance à la sortie
  if elle est inférieure à la valeur précédente then
    la définir comme la nouvelle distance
    actualiser le tas
  end if
  marquer le point comme considéré
end for
while le tas est non vide do
  extraire la tête du tas et la marquer comme calculée
  actualiser le tas
  for all points dans le voisinage de la tête do
    déterminer leur distance à la sortie
    if elle est inférieure à la valeur précédente then
      la définir comme la nouvelle distance
      actualiser le tas
    end if
    marquer le point comme considéré
  end for
end while
```

1.2 Détails

calcul d'une distance On considère que l'onde se propage de manière uniforme, donc la norme de son gradient est constante que l'on fixe à 1.

On note $x_{i,j}$ le point de coordonnées (i, j) et $x_{i-1,j}, x_{i+1,j}, x_{i,j-1}, x_{i,j+1}$ ses voisins. On note $D(x)$ la distance de tout point x.

Le calcul de la distance de $x_{i,j}$ se fait à partir des distances de ses voisins qui ont été considérés comme calculés, et repose sur l'approximation :

$$\left\|\vec{grad}(D)(x_{i,j})\right\|^2 \approx (max(D(x_{i,j})-D(x_{i-1,j}), D(x_{i,j})-D(x_{i+1,j})))^2+(max(D(x_{i,j})-D(x_{i,j-1}), D(x_{i,j})-D(x_{i,j+1})))^2$$

Comme on prend $\left\|\vec{grad}(D)(x_{i,j})\right\|^2 = 1$, on obtient en notant $d_h = min(D(x_{i-1,j}), D(x_{i+1,j}))$ et $d_h = min(D(x_{i,j-1}), D(x_{i,j+1}))$:
$1 = (D(x_{i,j}) - d_h)^2 + (D(x_{i,j}) - d_v)^2$
On déduit $D(x_{i,j}) = \frac{d_v - d_h + \sqrt{2-(d_v-d_h)^2}}{2}$

Erreur, on a un dh et un dv

optimisation à l'aide d'un tas Afin d'accéder rapidement au point non accepté de distance la plus faible, on stocke les points non acceptés dont la distance a été calculée dans un tas, un arbre binaire dans lequel la distance d'un nœud est plus faible que celle de ses fils. Il suffit alors d'extraire la tête de l'arbre pour obtenir le point recherché.

Le tas est actualisé au cours de l'exécution de l'algorithme à l'aide des procédures suivantes :
- Remonter, qui permet de rétablir le tas après que la valeur d'une distance ait été modifiée
- Insérer, qui permet d'ajouter un nouveau point au tas
- Extraire qui permet d'extraire la tête tout en conservant la structure de tas

1.3 Preuve et complexité

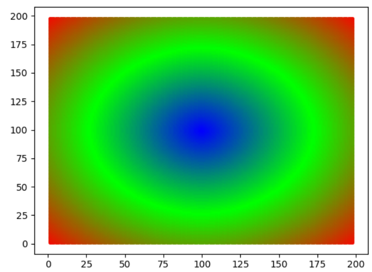
De même que pour l'algorithme de Dijkstra, la preuve repose sur l'invariant de boucle suivant : "les distances des points marqués comme calculés définitivement sont minimales".

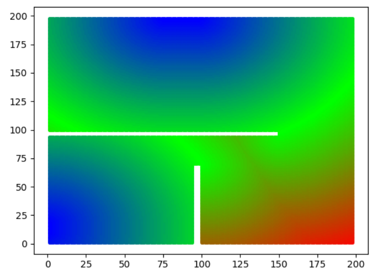
Si l'on considère les appels à la fonction de calcul de distance, alors l'algorithme a une complexité linéaire par rapport à l'aire de la grille.

Si l'on prend également en compte la comparaison de distances, utilisée par les procédures de gestion du tas qui ont une complexité linéaire par rapport à la taille de ce dernier, la complexité de l'algorithme passe en $O(n \log n)$ avec n le nombre de cases de la grille.

1.4 Résultats

Les deux images ci-dessus montrent les résultats obtenus pour une salle sans mur et avec une sortie au centre, et pour une salle comportant deux sorties et deux murs. On observe bien une propagation uniforme et cohérente.





2 Détermination de la vitesse réelle avec l'algorithme de Uzawa

Avec la carte des distances obtenue précédemment, on peut déterminer la vitesse souhaitée d'un individu en en prenant le gradient, et ainsi construire la vitesse souhaitée du groupe d'individu :
$V_s = (v_1, ..., v_N) \in \mathbb{R}^{2N}$

Pour éviter les collisions, la vitesse réelle V doit appartenir à l'ensemble
$Adm = \{V \in \mathbb{R}^{2N} | \forall (i,j) \in [\![1, N]\!]^2, h \, \langle (v_i - v_j) \mid e_{i,j} \rangle < D_{i,j} - R\}$

Ensemble des vitesses acceptées et acceptables

Avec $D_{i,j}$ la distance entre deux individus, R leur rayon, h le pas de temps de la méthode d'Euler et $e_{i,j}$ le vecteur unitaire reliant deux personnes.

On va donc projeter la vitesse souhaitée sur cet ensemble, en utilisant la méthode de Uzawa décrite dans [1].

2.1 Principe et préliminaires

L'ensemble des vitesses admissibles est convexe et fermé, donc on est assuré de l'existence et de l'unicité du projeté, que l'on cherche désormais à calculer.

Cela revient à trouver l'élément de Adm minimisant la distance à V_s. On ne peut pas appliquer la méthode du gradient usuellement utilisée pour minimiser une fonction, car elle nécessite de projeter à chaque étape sur l'ensemble Adm, ce que l'on cherche à réaliser. On va donc se ramener à un problème où l'on pourra utiliser la méthode du gradient.

Définition 1 (point selle)**.** *Soit f une application d'un produit cartésien $U \times M$ vers $\mathbb{R}$, on dit que (u, λ) est point selle de f si*
$f(v, \lambda) = \inf_{u \in U} f(u, \lambda) = \sup_{\mu \in M} f(v, \mu)$

On va montrer que le projeté de Vs sur Adm est le premier argument d'un point selle du lagrangien défini par :

$$L : \mathbb{R}^{2N} \times \mathbb{R}_+^{\frac{N(N-1)}{2}}, (v, \lambda) \mapsto \frac{1}{2}\|v - V_s\|^2 + \sum_{1 \leqslant i < j \leqslant N} \lambda_{i,j}(h \langle (v_i - v_j) \mid e_{i,j} \rangle - D_{i,j} + R)$$

On pourra alors chercher à trouver le second argument du point selle, en utilisant la méthode du gradient, car la projection sur $\mathbb{R}_+^{\frac{N(N-1)}{2}}$ est facile à réaliser.

Théorème 1. *Si (v, λ) est point selle de L, v est le projeté de V_s sur Adm*

Démonstration. Comme $L(v, \lambda) = \inf \left\{ L(v, \mu), \mu \in \mathbb{R}_+^{\frac{N(N-1)}{2}} \right\}$, pour tout μ dans $\mathbb{R}_+^{\frac{N(N-1)}{2}}$,

$$\sum_{1 \leqslant i < j \leqslant N} (\lambda_{i,j} - \mu_{i,j})(h \langle (v_i - v_j) \mid e_{i,j} \rangle - D_{i,j} + R) \geqslant 0$$

Donc en fixant tous les $\mu_{i,j} = \lambda_{i,j}$ sauf un que l'on fait tendre jusqu'à l'infini, on obtient

$\forall\, 1 \leqslant i < j \leqslant N, h \langle (v_i - v_j) \mid e_{i,j} \rangle - D_{i,j} + R \leqslant 0$ donc $v \in Adm$.
Et de même avec 0 à la place de l'infini, $\sum_{1 \leqslant i < j \leqslant N} \lambda_{i,j}(h \langle (v_i - v_j) \mid e_{i,j} \rangle - D_{i,j} + R) = 0$

Enfin, $\forall u \in Adm$,
$$\begin{aligned} \tfrac{1}{2}\|v - V_s\|^2 &= L(v, \lambda) \\ &\leqslant L(u, \lambda) \\ &\leqslant \tfrac{1}{2}\|u - V_s\|^2 + \sum_{1 \leqslant i < j \leqslant N} \lambda_{i,j}(h \langle (u_i - u_j) \mid e_{i,j} \rangle - D_{i,j} + R) \\ &\leqslant \tfrac{1}{2}\|u - V_s\|^2 \text{ donc } v \text{ est bien le projeté sur } Adm \end{aligned}$$
□

Théorème 2. *Si v est le projeté sur Adm, un point selle existe*

Démonstration. On utilisera les relations de Kuhn et Tucker, qui affirment que si J et $(\varphi_i)_{1 \geqslant i \geqslant m}$ sont des fonctions convexes différentiables sur un espace vectoriel E dans $\mathbb{R}$, si on note
$U = \{v \in E | \forall i \in [\![1, m]\!], \varphi_i(v) \leqslant 0\}$. Si u est un minimum de J par rapport à U et si les $(\varphi_i)_{1 \geqslant i \geqslant m}$ sont affines, alors il existe λ dans $\mathbb{R}_+^m$ tel que :

$$\begin{cases} dJ(v) + \sum_{i=1}^{m} \lambda_i d\varphi_i(v) = 0 & (1) \\ \sum_{i=1}^{m} \lambda_i \varphi_i(v) = 0 & (2) \end{cases}$$

De (2) on tire

$$\begin{aligned} \forall \mu \in \mathbb{R}^{\frac{N(N-1)}{2}}, L(v, \mu) &= \frac{1}{2}\|v - V_s\|^2 + \sum_{1 \leqslant i < j \leqslant N} \mu_{i,j}(h \langle (v_i - v_j) \mid e_{i,j} \rangle - D_{i,j} + R) \\ &\leqslant \frac{1}{2}\|v - V_s\|^2 = L(v, \lambda) \end{aligned}$$

Et (1) est une condition suffisante de minimum d'une fonction convexe, donc (λ, μ) est point selle. $\square$

2.2 Description et preuve de convergence

La méthode de Uzawa On cherche à trouver le second argument du point selle, c'est-à-dire trouver $\lambda \in \mathbb{R}_+^{\frac{N(N-1)}{2}}$ tel que $G(\lambda) = \sup \left\{ G(\mu), \mu \in \mathbb{R}_+^{\frac{N(N-1)}{2}} \right\}$, avec $G(\mu) = \inf\limits_{v \in \mathbb{R}^{2N}} L(v, \mu)$.

Pour cela, on va utiliser la méthode du gradient et construire par récurrence deux suites $(u_k)_{k\in\mathbb{N}}$ et $(\lambda_k)_{k\in\mathbb{N}}$ en prenant $\lambda_0 \in \mathbb{R}_+^{\frac{N(N-1)}{2}}$ de manière arbitraire, puis :

$$\begin{cases} u_k \text{ tel que } L(u_k, \lambda_k) = \inf\limits_{v\in\mathbb{R}^{2N}} L(v, \lambda_k) \\ (\lambda_{k+1})_i = max((\lambda_k)_i + \rho(h \langle ((u_k)_i - (u_k)_j) \mid e_{i,j} \rangle - D_{i,j} + R), 0) \end{cases}$$

Avec ρ une constante positive que l'on déterminera.

Théorème 3. *La suite $(u_k)_{k\in\mathbb{N}}$ converge vers v le projeté de V_s sur Adm*

Démonstration. L'application $\phi : \mathbb{R}^{2N} \to \mathbb{R}^{\frac{N(N-1)}{2}}, v \mapsto (h \langle (v_i - v_j) \mid e_{i,j} \rangle)_{1\leqslant i<j\leqslant N}$ est linéaire, notons C sa matrice dans les bases canoniques. On peut alors écrire $L(v, \mu) = \frac{1}{2}\|v - V_s\|^2 + \langle \mu | Cv \rangle - \langle \mu | D \rangle$. Avec $D = (D_{i,j} - R)_{1\leqslant i<j\leqslant N}$

Et donc avec (v, λ) un point selle de L, on obtient en différenciant selon v : $v - V_s + {}^tC\lambda = 0$

De plus, $\langle \mu - \lambda | Cv - D \rangle \leqslant 0$ donc $\langle \mu - \lambda | \lambda - (\lambda + \rho(Cv - D)) \rangle \geqslant 0$

D'où $\begin{cases} v - V_s + {}^tC\lambda = 0 \\ \lambda = P_+(\lambda + \rho(Cv - D)) \end{cases}$.

Et par construction, $(u_k, \lambda_k, \lambda_{k+1})$ vérifient les mêmes relations donc : $\begin{cases} u_k - v + {}^tC(\lambda_k - \lambda) = 0 \\ \|\lambda_{k+1} - \lambda\| \leqslant \|\lambda_k - \lambda + \rho C(u_k - v)\| \end{cases}$

En élevant au carré l'inégalité : $\|\lambda_{k+1} - \lambda\|^2 \leqslant \|\lambda_k - \lambda\|^2 + 2\rho \left\langle {}^tC(\lambda_k - \lambda) | u_k - v \right\rangle + \rho^2 \|C(u_k - v)\|^2$

Ainsi, $\|\lambda_{k+1} - \lambda\|^2 \leqslant \|\lambda_k - \lambda\|^2 - \rho(2 - \rho|||C|||^2)\|u_k - v\|^2$

Donc si $2 - \rho|||C|||^2 > 0$, la suite $(\|\lambda_k - \lambda\|^2)_{k\in\mathbb{N}}$ est décroissante donc converge.

Ainsi, comme $\|u_k - v\|^2 \leqslant \frac{\|\lambda_k - \lambda\|^2 - \|\lambda_{k+1} - \lambda\|^2}{\rho(2 - \rho|||C|||^2)}$, la suite $(u_k)_{k\in\mathbb{N}}$ converge vers v. $\square$

2.3 Implémentation informatique

J'ai utilisé le module `NumPy` pour représenter les vecteurs et matrices, dans un souci de performance : l'implémentation ne se fait pas avec des tableaux dynamiques donc on gagne en complexité spatiale et des fonctions de multiplication de multiplication, transposition, etc. sont disponibles et très efficaces en complexité temporelle.

Cartes des distances et des vitesses souhaitées Les distances sont calculées une seule fois au début du programme avec la FMM, et de même pour les vitesses souhaitées en chaque point de la carte, déterminées en approximant au second ordre le gradient des distances.

Dans la méthode de Uzawa On prend $\lambda_0 = 0$ ce qui permet de n'avoir qu'une itération s'il n'y a pas de chevauchement possible.

Pour déterminer une valeur du coefficient ρ, on utilise la majoration suivante :

$\forall 1 \leqslant i < j \leqslant N, |h \langle (v_i - v_j) \mid e_{i,j} \rangle|^2 \leqslant 4h^2\|v\|_\infty^2 \leqslant 4h^2\|v\|^2$

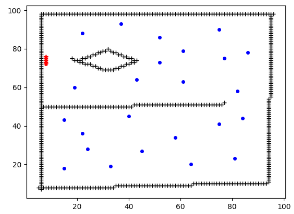
Donc $|||C|||^2 = \sum\limits_{1\leqslant i<j\leqslant N} |h \langle (v_i - v_j) \mid e_{i,j} \rangle|^2 \leqslant \frac{N(N-1)}{2} 4h^2\|v\|^2 < 2N^2h^2$

Ainsi, on prend $\rho = \frac{1}{N^2h^2}$, qui vérifie bien $2 - \rho|||C|||^2 > 0$.

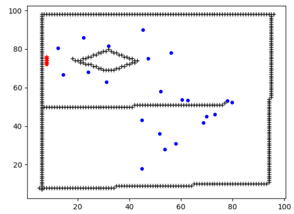
interface graphique En utilisant le module `Tkinter`, j'ai créé une classe permettant de dessiner à la souris le plan d'une salle avec des sorties, et de positionner les individus au départ de la simulation.

2.4 Résultats

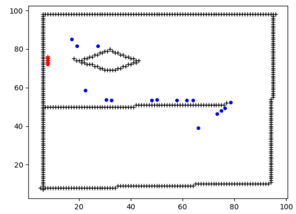
Les images suivantes ont été obtenues par deux simulations ayant duré environ deux heures chacune.



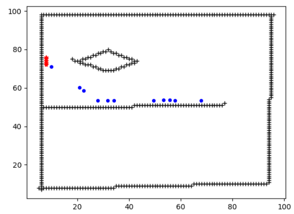
(a) t=1



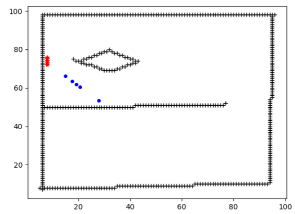
(b) t=150



(c) t=300

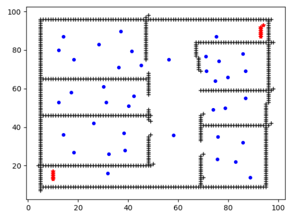


(d) t=450

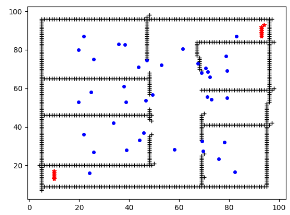


(e) t=650

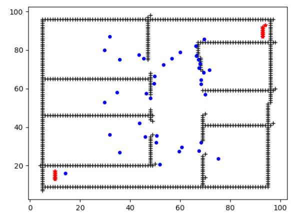
FIGURE 1 – Première simulation



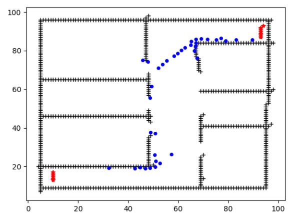
(a) t=1



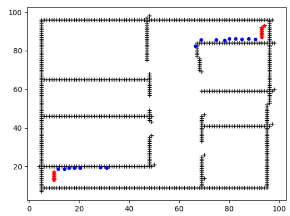
(b) t=50



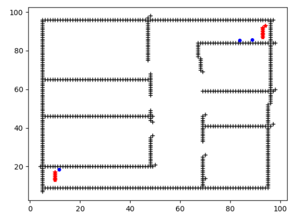
(c) t=100



(d) t=200



(e) t=400



(f) t=500

FIGURE 2 – Seconde simulation

Références

[1] Philippe G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson, 1990.

[2] Bertrand Maury et Juliette Venel. "Un modèle de mouvements de foule". In : *ESAIM : Proceedings* (2007).

[3] Mohamed Sylla et Boris Meden. *Méthode de Fast Marching, Calcul numérique de la distance à une Interface.* URL : `https://www.isima.fr/f4/projets2008/Fast_Marching_Sylla_Meden.pdf`.

A Codes carte des distances

```
##importations
import numpy as np
import matplotlib.pyplot as plt
from math import *

##exemples
mat=np.array([[(0,1) for i in range(200)]]+[[(0,1)]+[(0,2) for i in
range(198)]+[(0,1)] for j in range(198)]+[[(0,1) for i in range(200)]]) #1=limite
2=non calculé 3=calculé 4=considéré (mais non encore accepté)
for i in range(150):
    for j in range(95,100):
        mat[(i,j)]=(0,1)
for i in range(95,100):
    for j in range(70):
        mat[(i,j)]=(0,1)
##voisinnage
def voisinnage(M,c):
    a,b=c
    res=[]
    if M[(a-1,b)][1]!=1 and M[(a-1,b)][1]!=3:
        res.append((a-1,b))
    if M[(a+1,b)][1]!=1 and M[(a+1,b)][1]!=3:
        res.append((a+1,b))
    if M[(a,b-1)][1]!=1 and M[(a,b-1)][1]!=3:
        res.append((a,b-1))
    if M[(a,b+1)][1]!=1 and M[(a,b+1)][1]!=3:
        res.append((a,b+1))
    return(res)

##distance
def calcul_distance(M,c,pepins):
    a,b=c
    test=0
    if M[(a-1,b)][1]==3:
        if M[(a+1,b)][1]==3:
            x=min(M[(a-1,b)][0],M[(a+1,b)][0])
        else:
            x=M[(a-1,b)][0]
    elif M[(a+1,b)][1]==3:
        x=M[(a+1,b)][0]
    else:
        x=0
        test=2
    if M[(a,b-1)][1]==3:
        if M[(a,b+1)][1]==3:
            y=min(M[(a,b-1)][0],M[(a,b+1)][0])
        else:
            y=M[(a,b-1)][0]
    elif M[(a,b+1)][1]==3:
        y=M[(a,b+1)][0]
    else:
        y=0
        test=1
    if test==0:
        if (x-y)*(x-y)>2:
            pepins+=1
            if x<y:
                d=1+x
            else:
                d=1+y
        else:
            d=(x+y+sqrt(2-(x-y)*(x-y)))/2
    elif test==1:
        d=1+x
    else:
        d=1+y
    return(d,pepins)
```

```python
##gestion du tas
def remonter(t,i,pos):
    if i//2!=0 and t[i//2][0]>t[i][0]:
        t[i//2],t[i]=t[i],t[i//2]
        pos[t[i][1]],pos[t[i//2][1]]=pos[t[i//2][1]],pos[t[i][1]]
        remonter(t,i//2,pos)

def inserer(tas,v,d,pos):
    n=tas[0]
    pos[v]=n+1
    if n+1==len(tas):
        tas.append((d,v))
    else:
        tas[n+1]=(d,v)
    remonter(tas,n+1,pos)

def descendre(t,i,n,pos):
    if 2*i+1<n:
        if t[i][0]>t[2*i][0]:
            if t[2*i][0]<t[2*i+1][0]:
                t[i],t[2*i]=t[2*i],t[i]
                pos[t[i][1]],pos[t[2*i][1]]=pos[t[2*i][1]],pos[t[i][1]]
                descendre(t,2*i,n,pos)
            else:
                t[i],t[2*i+1]=t[2*i+1],t[i]
                pos[t[i][1]],pos[t[2*i+1][1]]=pos[t[2*i+1][1]],pos[t[i][1]]
                descendre(t,2*i+1,n,pos)
        elif t[i][0]>t[2*i+1][0]:
            t[i],t[2*i+1]=t[2*i+1],t[i]
            pos[t[i][1]],pos[t[2*i+1][1]]=pos[t[2*i+1][1]],pos[t[i][1]]
            descendre(t,2*i+1,n,pos)
    elif 2*i==n:
        if t[i][0]>t[2*i][0]:
            t[i],t[2*i]=t[2*i],t[i]
            pos[t[i][1]],pos[t[2*i][1]]=pos[t[2*i][1]],pos[t[i][1]]

##fonction principale
def carte_distance(M,sortie):
    pepins=0
    d=42
    f,g,h=np.shape(M)
    position=np.zeros((f,g),dtype=int)
    for c in sortie:
        M[c]=(0,3)
    tas=[0]
    for c in sortie:
        for v in voisinnage(M,c):
            d,pepins=calcul_distance(M,v,pepins)
            if M[v][1]==2:
                inserer(tas,v,d,position)
                tas[0]+=1
                M[v]=(d,4)
            elif d<M[v][0]:
                i=position[v]
                a,b=tas[i]
                tas[i]=(d,b)
                remonter(tas,i,position)
                M[v][0]=d
    while tas[0]!=0:
        n=tas[0]
        d,c=tas[1]
        tas[1]=tas[n]
        position[tas[1][1]]=1
        tas[0]-=1
        descendre(tas,1,n-1,position)
        M[c]=(d,3)
```

```
    for v in voisinnage(M,c):
        d,pepins=calcul_distance(M,v,pepins)
        if M[v][1]==2:
            inserer(tas,v,d,position)
            tas[0]+=1
            M[v]=(d,4)
        elif d<M[v][0]:
            i=position[v]
            a,b=tas[i]
            tas[i]=(d,b)
            remonter(tas,i,position)
            M[v][0]=d
print(pepins)
return(M,d)
```

B Codes affichage distances

```python
def echelle(color_begin, color_end, n_vals):
    r1, g1, b1 = color_begin
    r2, g2, b2 = color_end
    degrade = []
    etendue = n_vals - 1
    for i in range(n_vals):
        alpha = 1 - i / etendue
        beta = i / etendue
        r = r1 * alpha + r2 * beta
        g = g1 * alpha + g2 * beta
        b = b1 * alpha + b2 * beta
        degrade.append((r, g, b))
    return degrade
##
def affichage(m,sortie):
    M,dm=carte_distance(m,sortie)
    coul=echelle((0,0,1),(0,1,0),50)+echelle((0,1,0),(1,0,0),50)
    for i,c in enumerate(M):
        for j,d in enumerate(c):
            if d[1]!="lim":
                plt.plot(i,j,marker='.',markersize=5,color=coul[int(100*(d[0]/(dm+5))
)])
                if i>0 and j>0 and i<99 and j<99:
                    a,b=grad(M,(i,j))
                    plt.arrow(i,j,2*a,2*b)
    plt.show()
```

C Codes des fonctions principales

```
def grad(distances,c):
    a,b=c
    a,b=int(a),int(b)
    if distances[(a-1,b)][1]==1:
        if distances[(a+1,b)][1]==1:
            x=0
        else:
            x=distances[(a+1,b)][0]-distances[(a,b)][0]
    elif distances[(a+1,b)][1]==1:
        x=distances[(a,b)][0]-distances[(a-1,b)][0]
    else:
        x=(distances[(a+1,b)][0]-distances[(a-1,b)][0])/2
    if distances[(a,b-1)][1]==1:
        if distances[(a,b+1)][1]==1:
            y=0
        else:
            y=distances[(a,b+1)][0]-distances[(a,b)][0]
    elif distances[(a,b+1)][1]==1:
        y=distances[(a,b)][0]-distances[(a,b-1)][0]
    else:
        y=(distances[(a,b+1)][0]-distances[(a,b-1)][0])/2
    if x**2+y**2>0:
        n=sqrt(x**2+y**2)
        x,y=x/n,y/n
    return((x,y))
##
def G(i,j,q,N,h): #vecteurs unitaires entre les différents points multiplié par le
coefficient d'Euler
    res=np.zeros(2*N)
    a,b,c,d=q[2*i],q[2*i+1],q[2*j],q[2*j+1]
    norme=sqrt((a-c)**2+(b-d)**2)
    res[2*i],res[2*i+1]=h*(c-a)/norme,h*(d-b)/norme
    res[2*j],res[2*j+1]=h*(a-c)/norme,h*(b-d)/norme
    return(res)
##
def dis(i,j,q,r):  #distance entre deux points (en prenant en compte leur rayon)
    a,b,c,d=q[2*i],q[2*i+1],q[2*j],q[2*j+1]
    return(sqrt((a-c)**2+(b-d)**2)-2*r)
##
def vdist(q,r,N):  #toutes les distances
    res=[dis(i,j,q,r) for j in range(1,N) for i in range(j)]
    return(np.array(res))
##
def proj(u): #projeté sur R+
    n=len(u)
    res=np.array([0.]*n)
    for i,c in enumerate(u):
        if c>0:
            res[i]=c
    return(res)
##
def uzawa(us,q,h,N,r):
    if N>1:
        C=np.array([G(i,j,q,N,h) for j in range(1,N) for i in range(j)]) #matrice C
de la méthode (c'est bien ça j'ai vérifié)
        rho=1/(N**2*h**2)
#coefficient rho de la méthode
        lam=np.zeros(N*(N-1)//2)                                          #lambda de
la méthode
        condition=True
        dist=vdist(q,r,N)
        while condition:
            v=us-np.dot(np.transpose(C),lam)                              #us est la
vitesse souhaitée
            lam=proj(lam+rho*(np.dot(C,v)-dist))
            condition=False
            for c in vdist(q+h*v,r,N):
```

```
                if c<-r/10:
                    condition=True
    else:
        v=us
    return(v)
##
def vitesse_souhaitee(distance,vitesses,q,n):
    a=[]
    for i in range(n):
        a.append(vitesses[int(q[2*i])][int(q[2*i+1])])
    res=[]
    for c in a:
        a,b=c
        res.append(a)
        res.append(b)
    return(-1*np.array(res))
##
def gestion_graphique(mat,q,taille,N,sortie,vitesses):
    plt.clf()
    for i in range(1,taille):
        for j in range(1,taille):
            if mat[(i,j)][1]==1:
                plt.plot(i,j,marker='+',color='black')
    for i in range(N):
        plt.plot(q[2*i],q[2*i+1],marker='o',markersize=4,color='b')
    for c in sortie:
        a,b=c
        plt.plot(a,b,marker='*',color='r')
##
def simulation(mat,sortie,q0):
    q=[]
    for c in q0:
        a,b=c
        q.append(a)
        q.append(b)
    distances=carte_distance(mat,sortie)[0]
    N=len(q0)
    taille=len(mat)-1
    vitesses=[[(0,0)]*(taille+1)]+[[(0,0)]+[grad(distances,(i,j)) for j in 
range(1,taille)]+[(0,0)] for i in range(1,taille)]+[[(0,0)]*(taille+1)] # les trucs 
en + sont là pour les bords où la vitesse doit être nulle
    k=0
    while N!=0:
        k+=1
        gestion_graphique(mat,q,taille,N,sortie,vitesses)
        us=vitesse_souhaitee(distances,vitesses,q,N)
        v=uzawa(us,q,0.2,N,2)
        q=q+0.1*v
        sortis=[]
        for i in range(N):
            if (int(q[2*i]),int(q[2*i+1])) in sortie or (int(q[2*i])+1,int(q[2*i+1])) 
in sortie or (int(q[2*i]),int(q[2*i+1])+1) in sortie or 
(int(q[2*i])+1,int(q[2*i+1])+1) in sortie: #i est arrivé à la sortie
                sortis.append(i)
        if sortis!=[]:#il y a besoin d'actualiser q
            nouveau=[]
            for i in range(N):
                if i in sortis:
                    N-=1
                else:
                    nouveau.append(q[2*i])
                    nouveau.append(q[2*i+1])
            q=np.array(nouveau)
        plt.savefig("C:
\\Users\\guillaume\\Documents\\prepa\\TIPE\\déplacement\\onycrois{}".format(k))
```

D Codes de l’interface graphique

```
from tkinter import *

class CreerCarte:
    """interface graphique pour donner une carte"""


    def __init__(self):
        self.affichage=Tk()  #la fenetre graphique
        self.etat=0          #à quel stade on en est
        self.prems=True      #si le point que l'on place est le premier d'un segment
        self.points=[]       #les points délimitant des murs
        self.sorties=[]      #idem pour des sorties
        self.debut=(0,0)     #un premier point que l'on garde en mémoire
        self.sorties_finale=[]
        self.carte_finale=np.array([[(0,2) for i in range(200)] for j in range(200)])
        self.mecs=[]

        self.bouton=Button(self.affichage, text="passer à la sortie",
command=self.suivant)
        self.bouton.pack()

        self.canvas = Canvas(self.affichage, width=200, height=200,bg='lightblue')
        self.canvas.pack()

        self.canvas.bind("<Button-1>",self.point)


    def point(self,event):
        x,y=event.x,event.y
        if self.prems and self.etat!=2:
            self.debut=(x,y)
            self.prems=False
        elif self.etat!=2:
            if self.etat==0:
                self.canvas.create_line(self.debut,(x,y))
            else:
                self.canvas.create_line(self.debut,(x,y),fill='red')
            self.prems=True
        else:
            self.mecs.append((x,y))
            self.canvas.create_oval(x-2,y-2,x+2,y+2)
        if self.etat==0:
            self.points.append((x,y))
        elif self.etat==1:
            self.sorties.append((x,y))


    def suivant(self):
        if self.etat==0:
            self.bouton["text"]="placer les gens"
            self.etat=1
        elif self.etat==1:
            self.bouton["text"]="quitter"
            self.etat=2
        else:
            self.affichage.destroy()
            while self.points!=[]:
                d=self.points.pop()
                c=self.points.pop()
                ajout_ligne(self.carte_finale,c,d)
            while self.sorties!=[]:
                sortie_complete(self.sorties_finale,self.sorties)
            simulation(self.carte_finale,self.sorties_finale,self.mecs)

##
def ajout_ligne(mat,c,d):
    a,b=c
```

```python
    x,y=d
    if x<a:
        a,b,x,y=x,y,a,b
    if x==a:
        for j in range(b,y+1):
            mat[(a,j)][1]=1
    elif y-b>x-a:
        for j in range(b,y+1):
            mat[(int(a+(x-a)*(j-b)/(y-b)),j)][1]=1
    elif y-b>a-x:
        for i in range(a,x+1):
            mat[(i,int(b+(y-b)*(i-a)/(x-a)))][1]=1
    else:
        for j in range(y,b+1):
            mat[(int(a+(x-a)*(j-b)/(y-b)),j)][1]=1
##
def sortie_complete(s,out):
    d=out.pop()
    c=out.pop()
    a,b=c
    x,y=d
    if x<a:
        a,b,x,y=x,y,a,b
    if x==a:
        for j in range(b,y+1):
            s.append((a,j))
    elif y-b>x-a:
        for j in range(b,y+1):
            s.append((int(a+(x-a)*(j-b)/(y-b)),j))
    elif y-b>a-x:
        for i in range(a,x+1):
            s.append((i,int(b+(y-b)*(i-a)/(x-a))))
    else:
        for j in range(y,b+1):
            s.append((int(a+(x-a)*(j-b)/(y-b)),j))
```