Optimisation de l'évacuation en cas d'urgence

Jean Jouve

14 juin 2017

Table des matières

1 Étude des foules et évacuation **2**

2 Objectifs **2**

3 Simulation multi-agents **2**

4 Outils utilisés **3**

5 Description d'une salle **3**

6 Description des agents **4**

7 Modélisation du mouvement d'un agent **5**
7.1 Mouvement en fonction du voisinage 5
7.2 Mouvement en fonction du champ de vision 6
7.3 Mouvement par un champ vectoriel 8

8 Recherche d'optimisations grâce à la modélisation **12**

9 Conclusion **12**

A Listings **14**
A.1 `simulation.py` 14
A.2 `constructeur_simulation.py` 16
A.3 `espace.py` 21
A.4 `personne.py` 24
A.5 `test_point_suivre.py` (Modélisation du mouvement) 27
A.6 `space_hash.py` (Hachage de l'espace et Treillis) 40
A.7 `ecouteur.py` 46
A.8 `lieu_ferme.py` 47
A.9 `obstacle.py` 48
A.10 `representation.py` 49
A.11 `geometrie.py` 52
A.12 `base.py` 56

A.13 `fonctions_annexes.py` 61
A.14 `representation_categories.py` 61

1 Étude des foules et évacuation

L'étude du comportement des foules est un domaine important pour améliorer la sécurité lors des évènement publics ainsi que dans les bâtiments publics. Mais l'étude de ces foules par l'expérience coute chère car il est nécessaire de mobiliser un grand nombre d'individus, lors d'un simple festival de musique il peut y avoir plus de 7000 personnes. De plus les personnes prenant pars aux expériences ne seront pas dans les conditions réelles, ils ne seront donc pas paniqués par exemple, l'expérience peut alors produire des résultats erronés.

La conception de bonnes expérience permettant l'étude des foules étant difficile, l'étude se fait principalement grâce à des simulations se basant sur des vidéos et des descriptions d'évènements réelle ainsi que sur des résultats d'expériences exécuté avant le développement des simulations.

2 Objectifs

L'objectif du TIPE de mon trinôme fut de développer ne programme permettant d'étudier le mouvement d'une foule en cas d'évacuation dans différentes situation. Puis d'en déduire des optimisations possibles pour l'évacuation, particulièrement des optimisations concernant le plan d'évacuation et la disposition des obstacles.

Le TIPE s'est découpé en deux simulations à des échelles différentes, après le développement des simulation une étude des résultats produits par les simulations à été faite. Une des simulations se base sur un graphe à flux variant et est à l'échelle d'un bâtiment, l'autre simulation se base sur les simulations multi-agents et est à l'échelle d'une salle.

Je me suis concentré sur la modélisation du mouvement des agents dans la simulation multi-agents, je décrirai donc seulement les résultats de cette simulation.

3 Simulation multi-agents

Dans le domaine de la modélisation des foules les simulation multi-agents sont les simulations les plus présentes. Contrairement aux modélisation se basant sur des automates cellulaire et celles se basant sur le mouvement de particule qui modélise le mouvement de la foule dans son ensemble, les modélisations multi-agents modélise chaque agents individuellement le comportement de la foule étant un résultat du comportement de chacun de ces agents.

4 Outils utilisés

Nous avons codé les simulations en Python car c'est le seule langage connue par l'ensemble de notre trinôme, de plus Python viens avec un grand nombre de modules permettant de faciliter le développement de notre simulation.

Nous avons utilisé un module de la librairie standard pour manipulé des fichiers en JSON que nous avons utilisé comme fichiers de configurations pour nos simulations permettant ainsi de facilité l'étude de différentes situations.

Nous avons aussi utilisé le moteur physique en deux dimensions Pymunk, qui est une enveloppe autour du moteur physique Chipmunk codé en C, pour éviter de passer trop de notre temps voire tout notre temps sur la modélisation physique des obstacles et des agents. Nous avons utilisé en plus de Pymunk la librairie graphique Pygame pour afficher la simulation.

La structure de donnée utiliser par Pymunk pour ses calculs est une hiérarchie de volumes englobants, une structure couramment utilisé pour la détection de collisions dans un espace et le lancé de rayon – j'utilise cette disposition pour les lancé de rayons –, c'est un arbre binaire dont chaque nœuds est un volume qui englobe les volumes des nœuds enfants, les feuilles de cette arbre sont les objets de l'espace (Figure 1). Ainsi pour chercher les objets intersectant un objet donnée il suffit de rechercher récursivement dans les sous arbres dont le volume englobant racine intersecte l'objet donnée. Pymunk utilise des rectangles aligné aux axes des abscisses et des ordonnées couramment nommé AABB – axis-aligned bounding box – comme volume englobant. Pour avoir des recherches efficaces Pymunk équilibre l'arbre en minimisant la surface occupé par chaque nœuds.

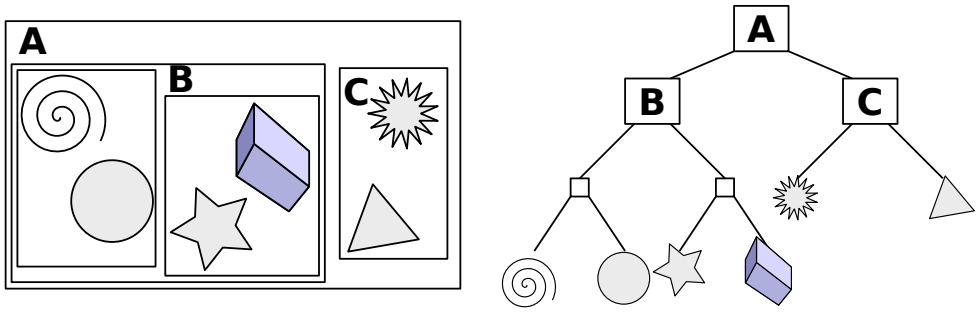

FIGURE 1 – Un exemple de BVH

Source: `https://en.wikipedia.org/wiki/Bounding_volume_hierarchy`

5 Description d'une salle

Les salles dans lesquelles sont étudiées le mouvement sont délimité par une AABB représentant les murs. Des trous dans les murs représentent les différentes

sorties. (Figure 2)

FIGURE 2 – Un exemple de salle, cette salle a une sortie en bas à droite

Les obstacles présent dans la salles – rangs d'une classe, pilier, etc... – ont d'abord était représentés par des AABB (Figure 3) car ce sont des formes très simples et donc ont permis de rapidement développer une modélisation correcte. Puis ces obstacles ont était représentés par des polygones convexes quelconques (Figure 3) pour atteindre une modélisation plus générale, la convexité ne pose pas de problèmes dans la généralité car tout polygones peut se décomposer en une union de polygones convexes – il existe une triangulation pour tout polygones simples – on peut donc représenté tout obstacle en accolant des polygones convexes ensembles mais ce n'est en générale pas nécessaire.

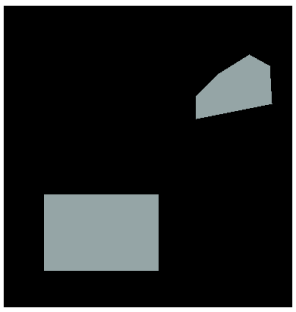

FIGURE 3 – Des exemples d'obstacles, une AABB à gauche et une polygone convexe quelconque à droite

6 Description des agents

Beaucoup de simplifications ont été faites pour les agents. Au mieux les agents devrait être représentés par des ellipses souples car en vus de dessus les agents ont grossièrement la forme d'ellipse et peuvent prendre plus ou moins de place selon la pression qui leurs est appliqué – ils plient les bras pour laisser de la place par exemple –. Mais une ellipse est une forme qui engendre des calculs de

géométrie compliqué et lent à exécuté d'autant plus si l'ellipse est souple. Nous avons donc choisi de représenter les agents par des disques rigides (Figure 4), cela permet d'avoir une simulation rapide sans perdre trop de réalisme, les disques étant une assez bonne approximation d'une ellipse et la différence de rayons entre les cas extrêmes et petite par rapport au rayon moyen.

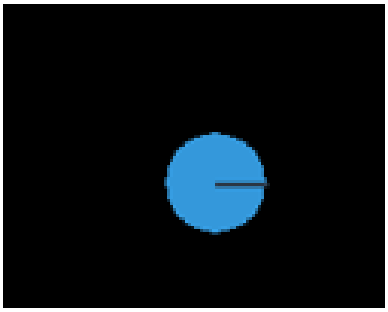

FIGURE 4 – Un agent

7 Modélisation du mouvement d'un agent

Mon objectif pour modéliser le mouvement d'un agent et de développer un algorithmes qui trouve la direction à suivre pour atteindre une certaine position étant donnée toutes les informations sur les obstacles et les autres agents tout en gardant un bon réalisme.

7.1 Mouvement en fonction du voisinage

En situation réelle les agents ne prennent pas en compte ce qui se trouve loin d'eux et ce qui ne se trouve pas dans leur champ de vision, le premier algorithme place donc des points autours de l'agent à une distance de un pas – trois fois le rayon de l'agent –on retire ensuite dans ces points l'ensemble des points étant inaccessibles, puis on choisi parmi les points restants le point le plus proche de la sortie. La distance caractéristique d'un pas permet aux points de bien prendre en compte du voisinage de l'agent tout en évitant les obstacles avant de rentrer en collision avec ce qui est plus réaliste

Le dernier point choisi est retenue après chaque choix de direction et l'agent continus à avancer dans cette direction jusqu'au prochain choix, l'algorithme fait un nouveau choix lorsque le dernier point choisi se retrouve bloqué.

Cette algorithme n'est pas du tout efficace pour amener l'agent en dehors de la sortie même si l'agent sort la plupart du temps, le chemin pris est loin d'être optimale, et contrairement à ce qui était recherché l'algorithme n'est pas du tout recherché, cela est du au manque de choix possibles de directions qui est limité au nombre de points et est du à la prise en compte de seulement le

voisinage beaucoup trop proche qui était forcé par le choix des points comme détection d'obstacle.

7.2 Mouvement en fonction du champ de vision

Pour remédier aux lacunes qu'as le choix de la direction par utilisation de points pour détecter les obstacles, j'ai choisi de détecter les obstacles grâce à des lancés de rayons, ceci permet de choisir parmi quasiment toutes les directions possibles et de prendre en compte la totalité du champ de vision de l'agent.

En plus du lancé de rayon on associe à chaque obstacle o de l'espace un ensemble d'obstacle $\mathcal{NC}(o)$ empêchant le contour de cette obstacle , c'est à dire en notant d la distance euclidienne, $\mathcal{O}$ l'ensemble des obstacles et r le rayon de l'agent on associe à l'obstacle o l'ensemble

$$\{o' \in \mathcal{O} | d(o, o') < 2r\}$$

Cette association est faite à l'aide d'une table de hachage à adressage ouvert. – ce sont les tables de hachages par défaut en python –

On calcule la distance $d(o, o')$ en utilisant le fait que cette distance est atteinte sur la frontière des obstacles, en notant $\mathcal{E}(o)$ l'ensemble des arrêtes de l'obstacle o et $\mathcal{V}(o)$ l'ensemble des sommets de l'obstacle o, on a

$$d(o, o') = \min\left\{ \begin{array}{l} \min\limits_{(v,v') \in \mathcal{V}(o) \times \mathcal{V}(o')} d(v, v'), \\ \min\limits_{(v,e') \in \mathcal{V}(o) \times \mathcal{E}(o')} d(v, e'), \\ \min\limits_{(e,v') \in \mathcal{E}(o) \times \mathcal{V}(o')} d(e, v') \end{array} \right\}$$

qui se détermine en $\mathcal{O}\left(|\mathcal{V}(o)||\mathcal{V}(o')|\right)$.

Il existe des algorithmes prenant $\mathcal{O}\left(\log\left(|\mathcal{V}(o)|\right) + \log\left(|\mathcal{V}(o')|\right)\right)$ de temps pour calculer cette distance car les obstacles sont convexes [1], mais la complexité quadratique n'est pas dérangeante car j'ai décidé de mettre en cache à l'aide d'une table de hachage les distances calculés et car le nombre de sommets par obstacles est assez faible dans la grande majorité des salles.

Pour choisir la direction l'algorithme exécute en premier un lancé de rayon vers la sortie et détecte ainsi si il y a un obstacle o sur droite de l'agent à la sortie, si il n'y a pas d'obstacle l'algorithme choisi la direction de la sortie comme direction à prendre, sinon l'algorithme recherche les rayons passant sur les bords des obstacle de $\mathcal{NC}(o)$ sans traversé aucun obstacle de $\mathcal{NC}(o)$, puis choisis le rayon ayant la plus petite distance angulaire avec le premier rayon lancé (Figure 5), il fait un choix glouton.

Pour décrire l'algorithme qui recherche les rayons pour le choix glouton on pose la fonction

$$f : \mathbb{R} \to \{1, 0\}$$

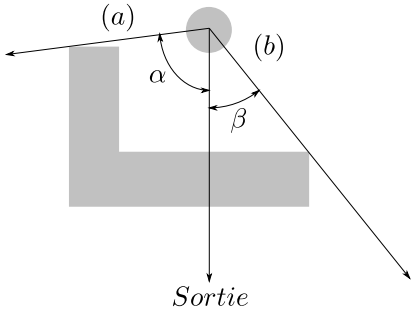

FIGURE 5 – Le choix des rayons se fait entre (a) et (b), comme $\alpha > \beta$ l'algorithme choisira (b) comme rayon à suivre

qui à $\theta \in \mathbb{R}$ associe 1 si il y a un obstacle de $\mathcal{NC}(o)$ dans cette direction et associe 0 sinon, ainsi l'objectif de l'algorithme est de trouver l'ensemble des angles θ tel que f change de valeur en θ c'est à dire l'ensemble

$$\mathcal{R} = \left\{\theta \middle| f\left(\theta^-\right) \neq f\left(\theta^+\right)\right\}$$

Pour calculer l'ensemble $\mathcal{R}$ il suffit de trouver les intervalles sur lesquels f est monotone pour pouvoir faire une dichotomie sur chacun de ces intervalles. L'algorithme trouve les intervalles par récurrence, il s'autorise d'omettre des rayons de $\mathcal{R}$ si le rayon omis est sur de ne pas être le rayon le plus proche de la sortie, il prend en entré un intervalle $I = [\alpha, \beta]$, en notant $m = \frac{\alpha+\beta}{2}$

- si $f(\alpha) = f(\beta)$ l'algorithme se relance sur $[\alpha, m]$ et $[m, \beta]$
- si $f(\alpha) \neq f(\beta)$ l'algorithme considère que f est monotone sur I – j'expliquerai cette considération – et lance une dichotomie sur I

Ainsi l'algorithme renvois $\mathcal{R} \cap I$, donc on lance l'algorithme avec pour entrée $I = [0, 2\pi]$ on obtient ainsi $\mathbb{R}$

Lorsque l'on a $f(\alpha) \neq f(\beta)$, f n'est pas forcément monotone, on pourrait se retrouver dans le cas de Figure 6, on a $o', o'' \in \mathcal{NC}(o)$ alors l'agent ne peut pas passer entre o' et o'' on peut donc changer f tel que $f(\theta) = 1$ entre o' et o'', on peut ainsi considérer f monotone sur $[\alpha, \beta]$. Ce changement dans f n'est pas fait dans l'implémentation de l'algorithme car je me suis rendu compte de cette propriété trop tard.

L'algorithme utilisant les lancés de rayons rend le mouvement beaucoup plus réaliste (Figure 7) et permet la sortie de la plupart des agents mais certains agents peuvent rester bloqué dans certaines configurations de salle (Figure 8), en fait tout algorithme non informé, ne stockant aucune information, utilisant un champ de vision, une fonction $f : \mathbb{R} \to \mathbb{N}$ tel que $f(\theta)$ indique l'obstacle dans la direction angulaire θ, peut se retrouver bloqué malgré l'existence d'une sortie.

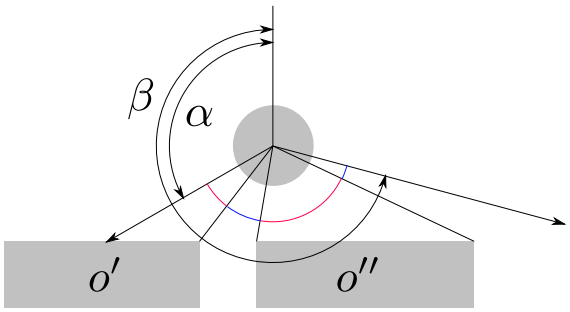

FIGURE 6 – Cas particulier de la situation $f(\alpha) = f(\beta)$, le bleu représentant les intervalles où $f(\theta) = 0$ et le rouge les intervalles où $f(\theta) = 1$

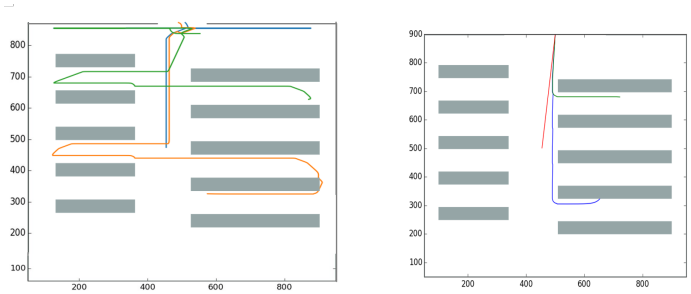

FIGURE 7 – Trajectoire de trois agents avec l'algorithme utilisant le voisinage (à droite) et avoir l'algorithme utilisant le lancé de rayon (à gauche)

7.3 Mouvement par un champ vectoriel

On a vu qu'un agent peut se retrouver dans une situation où un algorithme de lancé de rayon ne peut l'en sortir car l'algorithme n'est pas informé. Pour régler ce problème j'ai supposé que l'agent avait bonne connaissance de la salle lors de l'évacuation il connais donc le chemin à prendre à l'avance.

Cela m'à amené à créer un champ vectoriel $\vec{C} : S \to \mathbb{R}^2$ sur l'espace S de la salle tel que $\vec{C}(x, y)$ indique la direction à prendre par un agent étant à la position (x, y).

Pour la première implémentation de ce champ, j'ai utilisé un hachage de l'espace H comme structure de donnée pour stocker le champ. Un hachage de l'espace est une découpe de l'espace en cellules stocké dans un tableau à deux dimensions qui associe à un point (x, y) de l'espace la cellule dans laquelle le point (x, y) se trouve en temps constant [2]. On accède en temps constant à une cellule car les indices (i, j) de la cellule peuvent être récupérés grâce à la formule

$$j = \left\lfloor \frac{x - H.x}{t} \right\rfloor$$

où $H.x$ est l'abscisse du coin bas droit de l'hachage de l'espace et t et la taille

FIGURE 8 – Blocage de tout algorithme non informé utilisant un champ de vision

d'une cellule (Figure 9), la formule pour i est analogue.

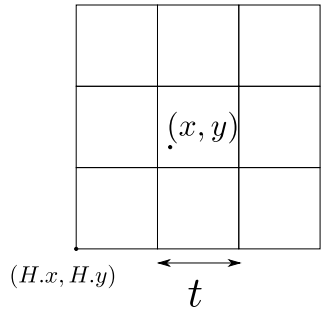

FIGURE 9 – Un exemple d'hachage de l'espace

Ainsi l'algorithme doit remplir chaque cellules de l'hachage de l'espace par un vecteur indiquant la direction à prendre par un agent se trouvant dans cette cellule. Le remplissage de l'hachage de l'espace s'est fait par un parcours en largeur où j'ai considérer que deux cellules sont voisines si les cellules se touches – deux cellules ayant des sommets se touchants se touchent – et si ont peut aller d'une cellule à une autre sans rencontrer d'obstacle, pour tester cela l'algorithme fait un lancer de rayon entre les deux cellules et indique qu'il est possible d'aller de l'une à l'autre si aucun obstacle n'as été touché par le lancer de rayon. Lorsque le parcours en largeur traite une cellule il regarde toute les voisines et assigne à chacune des cellules un vecteur dirigé vers la cellule état traité, ainsi comme le parcours en largeur traite d'abord les cellules proches des sortie le champ dirige progressivement les agents vers la sortie (Figure 10).

On peut voire dans Figure 10 que le parcours en largeur ne donne pas le plus cours chemin, cela est du à l'aspect continue de l'espace et l'aspect discret du parcours en largeur ainsi qu'au fait que la distance entre deux cellules n'est pas toujours 1 mais diffère, un meilleur champ aurait pu être obtenue en utilisant un algorithme du plus cours chemin tel l'algorithme de Dijkstra avec des arrêtes de poids 1 et $\sqrt{2}$ ce qui aurait permis de mieux se rapprocher du plus cours chemin réelle au plus – 8% plus long [3] –, mais un manque de temps à empêché

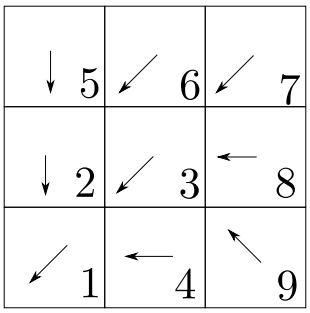

FIGURE 10 – Un exemple de parcours, les cellule sont numéroté par l'ordre de traitement

d'implémenter cela.

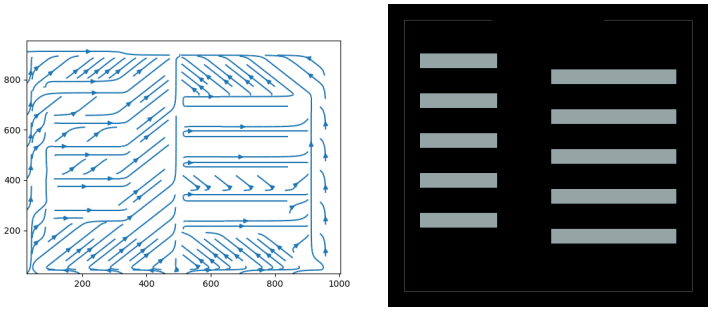

FIGURE 11 – Un champ vectorielle et la salle sur lequel il a été créé

Le premier algorithme de champ vectorielle fait bien sortir les agents mais le mouvement des agents n'est pas très réaliste car les agents suivent un chemin sur un quadrillage (Figure 11), pour remédier à cela le second algorithme calcule un champ de gradient dérivé d'un champ de scalaire qu'il construit. Je fait en sorte que le gradient dépende plus de la position de l'agent que de la case où se trouve l'agent pour atteindre un meilleur réalisme.

Cette algorithme utilise une structure de donnée que l'on nommera treillis très similaire à celle de l'algorithme précédent, c'est un quadrillage de l'espace qui associe à chaque croisement du quadrillage une valeur, le treillis est tel que l'on peut récupérer en temps constant les valeurs étant aux coins d'une case dans laquelle se trouve un point (Figure 12).

Pour remplir le treillis on procède de la même façon que pour l'algorithme précédent, mais au lieu d'assigner des vecteurs le parcours en largeur assigne à chaque intersection du quadrillage l'opposé de la distance à la sortie la plus proche, ainsi le gradient donnera la direction vers la sortie la plus proche.

Pour calculer le gradient en un point (x, y) de l'espace, l'algorithme créer un champ scalaire définis sur tout l'espace de la salle S en interpolant les valeurs du treillis, puis l'algorithme dérive le champ scalaire en (x, y) pour avoir le gradient.

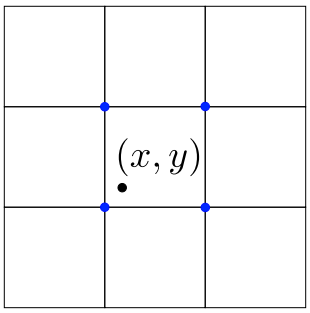

FIGURE 12 – Le treillis permet de récupérer les valeurs aux points bleus à partir du point

Ces deux étapes sont exécutées en une seule en dérivant directement la formule d'interpolation. L'utilisation de l'interpolation permet de donner la dépendance recherché du gradient à la position.

La formule d'interpolation utilisé et la formule d'interpolation bilinéaire [4] qui est une formule quadratique en x et y, elle permet d'avoir un champ continus en tous points de l'espace et dérivable sur l'intérieur – d'un point de vue topologique – des cases du quadrillage. La formule d'interpolation bicubique [5] permet d'avoir un champ dérivable en tout point mais elle crée un comportement étrange chez les agents que je n'ai pas réussi à expliquer, elle n'a donc pas été retenue.

Le champ de gradient n'as pas le manque de réalisme que le premier algorithme a (Figure 14 mais pose un problème, les agents restent bloqués au niveau des coins des obstacles (Figure 13). Cela est surement du au remplissage par un parcours en largeur qui ne donne pas la distance exacte aux sorties.

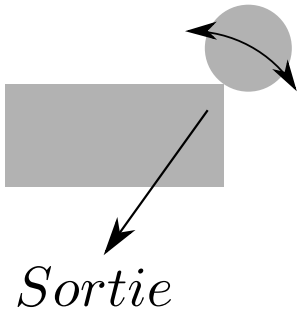

FIGURE 13 – Mouvement de l'agent proche d'un coin avec l'algorithme utilisant un gradient

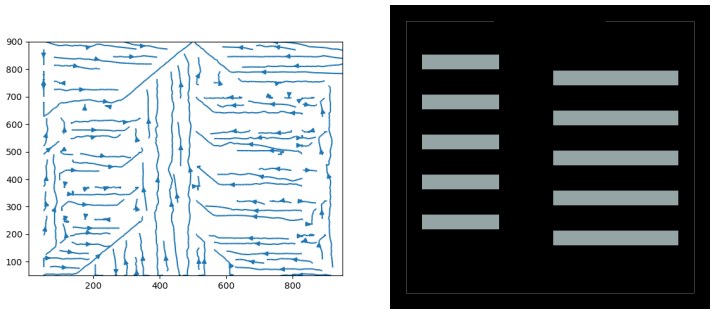

FIGURE 14 – Champ de gradient à gauche obtenue avec l'algorithme dans la salle à droite

8 Recherche d'optimisations grâce à la modélisation

Pour rechercher les optimisations possibles, la modélisation est utilisé avec l'algorithme utilisant les lancés de rayons, et si il y a des obstacles dans la salles les obstacles sont suffisamment espacé pour ne pas rentrer dans un blocage. Dans ces conditions l'algorithme utilisant les lancés de rayons donne un mouvement plus réaliste que l'algorithme utilisant le champ de gradient.

La temps qu'a pris la modélisation du mouvement des agents nous a empêché de passer beaucoup de temps sur la recherche d'optimisation possibles, nous nous sommes donc concentré sur la fluidification débit par le placement d'un obstacle en face de la sortie. Cette optimisation est souvent mentionné dans la littérature scientifique [7].

Nous avons déterminer la distribution du débit moyen de deux salles, une vide et une autre vide avec un obstacle devant la porte censé fluidifier le mouvement (Figure 15). On voit clairement que l'obstacle devant la porte réduit considérablement le débit. Ce résultat ne veut pas dire que l'obstacle ne fluidifie pas en situation réelle, en effet le ralentissement que génère l'obstacle est surtout du au manque de finesse de notre modélisation. Notre simulation ne prend pas suffisamment en compte les interactions humaine tel que la tendance à éviter autrui que les personnes ont (, qui est la raison du manque de fluidification.

9 Conclusion

Les limites de notre modélisation à permis de mettre en valeur certain points importants dans l'implémentation d'une simulation du mouvement de foule tel que l'intéraction entre les agents.

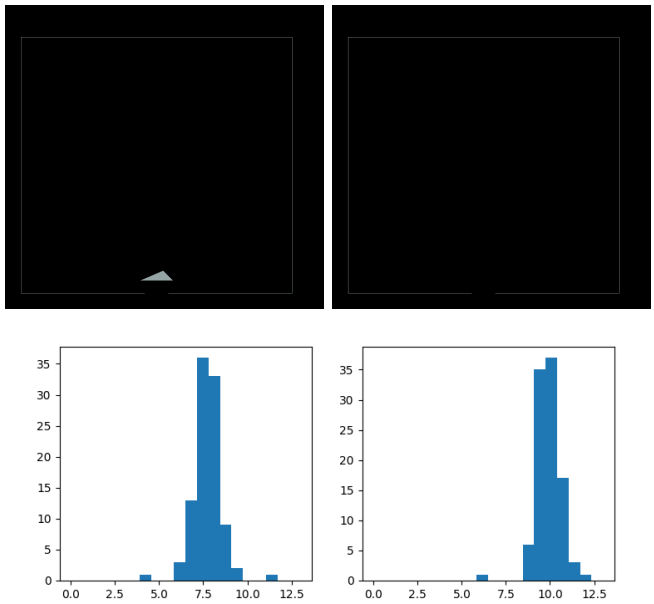

FIGURE 15 – répartition des débit moyen d'une simulation sur 100 simulation pour chaque salle

Références

[1] Yang Cheng-lei, Qi Meng, Meng Xiang-xu, Li Xue-qing, Wang Jia-ye, *A new fast algorithm for computing the distance between two disjoint convex polygons based on Voronoi diagram*, Journal of Zhehiang University SCIENCE A, 2006.

[2] Tristam MacDonald, *Spatial Hashing*, `https://www.gamedev.net/resources/_/technical/game-programming/spatial-hashing-r2697`, 2009

[3] Alex Nash, *Any-Angle Path Planning*, `http://idm-lab.org/bib/abstracts/papers/dissertation-nash.pdf`, 2012

[4] Wikipedia, *Interpolation bilinéaire*, `https://fr.wikipedia.org/wiki/Interpolation_bilin%C3%A9aire`

[5] Wikipedia, *Interpolation bicubique*, `https://fr.wikipedia.org/wiki/Interpolation_bicubique`

[6] Daniel Flower, *Crowd Simulation for Emergency Response Planning*

[7] Constantin Theos, *Modélisation du mouvement des personnes lors de l'évacuation d'un bâtiment à la suite d'un sinistre*, `https://tel.archives-ouvertes.fr/file/index/docid/523176/filename/1994TH_THEOS_C_NS20040.pdf`, 1994

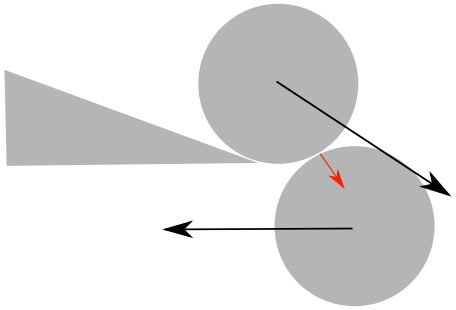

FIGURE 16 – Situation où un agent ralentit un autre agent lors de sa sortie au niveau d'un obstacle, cette situation à lieu car les agents n'essaient pas d'éviter les autres agents comme c'est le cas en situation réelle

A Listings

A.1 simulation.py

```
1  import time
2  import base
3
4  class Simulation(object):
5      '''S'occupe d'ajouter les écouteurs aux personnes de l'espace
6      et de mettres à jour tous les éléments nécessaire à la
7      simulation lorsqu'il lui est demandé
8
9      creer_ecouteur: une fonction prenant une personne en entree et
10         renvoyant un ecouteur associé à cette personne
11
12     '''
13
14     AUCUN = 0x0
15     ARRET = 0x1
16     TOGGLE_PAUSE = 0x2
17
18     def __init__(self, espace, nombre_mise_a_jour_par_seconde, creer_ecouteur):
19         self.espace = espace
20         self.mise_a_jour_par_seconde = nombre_mise_a_jour_par_seconde
21         self.ecouteurs = []
22         self.sources = []
23         self.action_mise_a_jour = lambda simulation: None
24         self.en_marche = False
25
26         self.creer_ecouteur = creer_ecouteur
27
```

```
28          self.espace.rappelle_personne_ajoute = base.EnsembleRappelle()
29          self.espace.rappelle_personne_ajoute.ajouter(self.ajouterEcouteurPourPersonne)
30
31      @property
32      def rappelle_personne_ajoute(self):
33          return self.espace.rappelle_personne_ajoute
34
35      @property
36      def ensemble_personnes(self):
37          return self.espace.ensemble_personnes
38
39      def mettreAJour(self):
40          self.temps_depuis_lancement += 1 / self.mise_a_jour_par_seconde
41          self.espace.avancer(1 / self.mise_a_jour_par_seconde)
42          for ecouteur in self.ecouteurs:
43              ecouteur.ecouter(self.temps_depuis_lancement)
44          self.mettreAJourSource()
45
46      def ajouterEcouteurPourPersonne(self, personne):
47          self.ecouteurs.append(self.creer_ecouteur(personne))
48
49      def gererActionExterieur(self):
50          commande = self.action_mise_a_jour(self)
51          self.executerCommande(commande)
52
53      def mettreAJourSource(self):
54          for source in self.sources:
55              source.mettreAJour(self.temps_depuis_lancement)
56
57      def executerCommande(self, commande):
58          if commande & Simulation.ARRET:
59              self.en_marche = False
60          if commande & Simulation.TOGGLE_PAUSE:
61              self.en_pause = not self.en_pause
62
63      def lancer(self):
64          self.debut_lancement = time.time()
65          self.temps_depuis_lancement = 0
66          self.en_marche = True
67          self.en_pause = False
68          while self.en_marche:
69              self.gererActionExterieur()
70              if self.en_pause:
71                  continue
72              self.mettreAJour()
```

A.2 constructeur_simulation.py

```
from lieu_ferme import LieuFerme
from personne import Personne
from obstacle import ObstacleRectangulaire, ObstacleCirculaire
from obstacle import ObstaclePolygonale
import base
from ecouteur import EcouteurPersonne
from espace import Espace
from pymunk.vec2d import Vec2d
from random import random, randint
from source_personne import Source
from math import sqrt
from simulation import Simulation

#TODO: éviter les constante tout à fait arbitraire

class ConstructeurSalle(object):

    def __init__(self, donnees_simulation):
        self.donnees_simulation = donnees_simulation

        self.espace = Espace()
        self.type = self.donnees_simulation['type']

        self.ajouterLieuFerme(
            self.espace,
            self.donnees_simulation['personnes']['zone_apparition'],
            **self.donnees_simulation['lieu_ferme']['salle'])

        if self.type == "salle_de_classe" :

            self.ajouterRangs(
                self.espace,
                self.donnees_simulation['personnes']['zone_apparition'],
                **self.donnees_simulation['obstacles']['rangs'])

        if self.type == "salle_en_T":

            self.ajouterFormeT(
                self.donnees_simulation['obstacles']['particulier']['rectangles'],
                self.donnees_simulation['personnes']['zone_apparition'],
                **self.donnees_simulation['lieu_ferme']['salle_couloir'])

        if self.type == "salle_en_Y":

```

```
45              self.ajouterFormeY(
46                  self.donnees_simulation['obstacles']['particulier']['polygones'],
47                  self.donnees_simulation['personnes']['zone_apparition'],
48                  **self.donnees_simulation['lieu_ferme']['salle_couloir'])
49
50
51          self.ajouterObstacles(
52              self.espace,
53              self.donnees_simulation['obstacles']['particulier'])
54
55
56
57      def ajouterLieuFerme(self,
58              espace,
59              zone_apparition=None,
60              salle_hauteur=None,
61              salle_largeur=None,
62              porte_largeur=None,
63              porte_position=None):
64
65          zone_apparition.update({'x_min' :  50})
66          zone_apparition.update({'x_max' : 50 + salle_largeur })
67          zone_apparition.update({'y_min' :  50 })
68          zone_apparition.update({'y_max' :  50 + salle_hauteur })
69
70
71          espace.ajouterLieuFerme(LieuFerme(
72              self.donnees_simulation['lieu_ferme']['porte'],
73              salle_largeur,
74              salle_hauteur,
75              Vec2d(50, 50)))
76
77      def ajouterFormeT(self,
78              rectangles,
79              zone_apparition,
80              largeur_horizontale=None,
81              largeur_verticale=None):
82
83          largeur_couloir = largeur_horizontale
84          hauteur_obstacle = largeur_verticale
85          largeur_obstacle = (self.espace.lieu_ferme.largeur - largeur_couloir)/2
86
87          coin_inferieur1 = [50,50]
88          coin_inferieur2 = [50 + largeur_couloir + largeur_obstacle  ,50]
89
90          rectangles.append({
```

```
            "largeur" : largeur_obstacle,
            "hauteur" : hauteur_obstacle,
            "position" : coin_inferieur1})
        rectangles.append({
            "largeur" : largeur_obstacle,
            "hauteur" : hauteur_obstacle,
            "position" : coin_inferieur2 })

        zone_apparition.update({'x_min' :  50 + largeur_obstacle})
        zone_apparition.update({'x_max' :  coin_inferieur2[0]})
        zone_apparition.update({'y_min' :  50})
        zone_apparition.update({'y_max' :  (50 + hauteur_obstacle)*(2/3)})



    def ajouterFormeY(self,
            polygones,
            zone_apparition,
            largeur_horizontale=None,
            largeur_verticale=None):

        #cf ficher annexe
        d = self.espace.lieu_ferme.largeur
        l = self.espace.lieu_ferme.hauteur
        a = largeur_horizontale
        b = largeur_verticale
        c = (d-a)/2
        y = 0
        x = sqrt(a**2 - y**2)
        h = ((d-2*x)/2) * sqrt(2)
        k = sqrt(h**2 - ((d-2*x)/2)**2)

        origine = [50,50]
        bordGauche = [[0,0],[c,0],[c,b],[0,b+c]]

        polygones.append({'position' : origine, 'sommets' : bordGauche})

        bordDroit = [[c+a,0],[d,0],[c+a,b],[d,b+c]]

        polygones.append({'position' : origine, 'sommets' : bordDroit})

        triangle = [[x,l],[d-x,l],[d/2,l-k]]

        polygones.append({'position' : origine, 'sommets' : triangle})

        zone_apparition.update({'x_min' : 50 + c} )
```

```
137             zone_apparition.update({'x_max' : 50 + c+a} )
138             zone_apparition.update({'y_min' : 50} )
139             zone_apparition.update({'y_max' : (50 + b)*(2/3)})
140 
141 
142         def ajouterObstacles(self, espace, particulier):
143             self.ajouterObstaclesParticulier(espace, particulier)
144 
145         def ajouterObstaclesParticulier(self, espace, obstacles):
146             for obstacle in obstacles['rectangles']:
147                 espace.ajouterObstacle(ObstacleRectangulaire(**obstacle))
148             for obstacle in obstacles['cercles']:
149                 espace.ajouterObstacle(ObstacleCirculaire(**obstacle))
150             for obstacle in obstacles['polygones']:
151                 espace.ajouterObstacle(ObstaclePolygonale(**obstacle))
152 
153 
154         def ajouterRangs(self,
155                 espace,
156                 zone_apparition,
157                 largeur_gauche=None,
158                 largeur_droit = None,
159                 hauteur = None,
160                 distance_intermediaire=None,
161                 distance_au_mur=None,
162                 position_debut_gauche=None,
163                 position_debut_droit=None):
164 
165             position_gauche_y = position_debut_gauche
166             position_droit_y = position_debut_droit
167 
168             zone_apparition.update({'x_min' :  50})
169             zone_apparition.update({'x_max' :  50 + self.espace.lieu_ferme.largeur})
170             zone_apparition.update({
171                 'y_min' :  50 +
172                     min(position_debut_gauche, position_debut_droit) + hauteur})
173             zone_apparition.update({'y_max' :  50 + self.espace.lieu_ferme.hauteur})
174 
175             #on ajoute les ranges de gauche
176             while position_gauche_y + 50 <= self.espace.lieu_ferme.hauteur :
177                 position_gauche = 50 + distance_au_mur, position_gauche_y
178 
179                 obstacle_gauche = ObstacleRectangulaire(
180                     hauteur = hauteur,
181                     largeur = largeur_gauche,
182                     position = position_gauche)
```

```
183
184             espace.ajouterObstacle(obstacle_gauche)
185
186             position_gauche_y += distance_intermediaire + hauteur
187
188
189         #on ajoute les rangs à droite
190         while position_droit_y + 50 <= self.espace.lieu_ferme.hauteur :
191             position_droit_x = (50 + self.espace.lieu_ferme.largeur
192                 - largeur_droit - distance_au_mur
193             position_droit = position_droit_x, position_droit_y
194
195             obstacle_droit = ObstacleRectangulaire(
196                 hauteur = hauteur,
197                 largeur = largeur_droit,
198                 position = position_droit)
199             espace.ajouterObstacle(obstacle_droit)
200
201             position_droit_y += distance_intermediaire + hauteur
202
203         zone_apparition.update({
204             'y_max' :
205                 50 + min(position_droit_y, position_gauche_y)
206                 - distance_intermediaire - hauteur})
207
208
209 class ConstructeurSimulation(object):
210
211     def __init__(self, donnees_simulation, action_sortie):
212         constructeur_salle = ConstructeurSalle(donnees_simulation)
213
214         creer_ecouteur = lambda personne: EcouteurPersonne(personne, action_sortie)
215
216         self.simulation = Simulation(
217             constructeur_salle.espace,
218             donnees_simulation['mise_a_jour_par_seconde'],
219             creer_ecouteur)
220
221         self.ajouterPersonnes(
222             nombre=donnees_simulation['personnes']['nombre'],
223             **base.fusioner_dictionnaires(
224                 donnees_simulation['personnes']['caracteristiques'],
225                 donnees_simulation['personnes']['zone_apparition']))
226
227         self.construireSources(
228             donnees_simulation['personnes']['sources'],
```

```
229             **donnees_simulation['personnes']['caracteristiques'])
230
231     def ajouterPersonnes(self,
232             nombre=0,
233             rayon_min = 30,
234             rayon_max = 30,
235             masse_surfacique = 1.8,
236             y_min=None,
237             y_max=None,
238             x_min=None,
239             x_max=None):
240
241         for _ in range(nombre):
242             personne = Personne(
243                 masse_surfacique,
244                 randint(rayon_min, rayon_max),
245                 Vec2d(
246                     x_min + random()*(x_max-x_min),
247                     y_min + random()*(y_max-y_min)),
248                 self.simulation.espace)
249
250             self.simulation.espace.ajouterPersonne(personne)
251
252     def construireSources(self,
253             liste_sources,
254             rayon_min=30,
255             rayon_max=30,
256             masse_surfacique=1.8):
257
258         for source in liste_sources:
259             self.simulation.sources.append(Source(
260                 self.simulation.espace,
261                 source['position'],
262                 source['periode'],
263                 rayon_min,
264                 rayon_max,
265                 masse_surfacique))
266
```

A.3 espace.py

```
1  import pymunk
2  from obstacle import ObstacleRectangulaire, OsbtacleSegment
3  from representation_categories import RepresentationCategorie, avoirMasqueSansValeur
4  from personne import Personne
5  import time
```

```
6  import geometrie
7  from pymunk import Vec2d
8
9
10 class Espace(pymunk.Space):
11
12     DIRECTIONS = [ Vec2d(0, -1), Vec2d(-1, 0), Vec2d(0, 1), Vec2d(1, 0) ]
13
14     def __init__(self):
15         super().__init__()
16
17         self.lieu_ferme = None
18         self.ensemble_obstacle = []
19         self.ensemble_personnes = []
20         self.ensemble_murs = []
21
22         #Pour eviter les calculs répété de distances entre des obstacles
23         self.calculateur_distance = geometrie.CalculateurDistanceAvecCache()
24
25         self.rappelle_personne_ajoute = lambda personne: None
26
27     def avancer(self, delta):
28         self.step(delta)
29
30         for personne in self.ensemble_personnes:
31             personne.update()
32
33     def avoirDistanceEntre(self, obstacle1, obstacle2):
34         return self.calculateur_distance.avoirDistanceEntre(
35             obstacle1,
36             obstacle2)
37
38     def peutPasserEntre(self, rayon, obstacle1, obstacle2):
39         return (
40             self.calculateur_distance.avoirDistanceEntre(
41                 obstacle1,
42                 obstacle2)
43             > rayon)
44
45     def cercleEstEnDehorsDeLieuFerme(self, position, rayon):
46         return any(map(self.lieu_ferme.pointEstAExterieur,
47             map(lambda direction: position + rayon * direction, Espace.DIRECTIONS)))
48
49     def avoirFiltre(self, ignorer_personne):
50         if ignorer_personne:
51             filtre = pymunk.ShapeFilter(mask=avoirMasqueSansValeur(
```

```
52                  pymunk.ShapeFilter.ALL_MASKS,
53                  RepresentationCategorie.PERSONNE.value))
54          else:
55              filtre = pymunk.ShapeFilter()
56          return filtre
57
58      def avoirInfoSurLancerRayon(self, debut, fin, ignorer_personne=True):
59          filtre = self.avoirFiltre(ignorer_personne)
60          epaisseur_rayon = 1
61
62          return self.segment_query_first(debut, fin, epaisseur_rayon, filtre)
63
64      def avoirInfoPoint(self, point, max_distance, ignorer_personne=True):
65          filtre = self.avoirFiltre(ignorer_personne)
66          return self.point_query_nearest(point, max_distance, filtre)
67
68      def pointEstDansObstacle(self, point):
69          return (self.avoirInfoPoint(point, 0) is not None
70              or self.lieu_ferme.pointEstAExterieur(point))
71
72      def ajouterPersonne(self, personne):
73          self.ensemble_personnes.append(personne)
74          self.add(personne.corps, personne)
75          self.rappelle_personne_ajoute(personne)
76
77      def ajouterObstacle(self, obstacle):
78          self.ensemble_obstacle.append(obstacle)
79          self.add(obstacle.corps, obstacle)
80
81      def recupererSommetsPorte(self, porte):
82          mur = self.lieu_ferme.avoirCote(porte['mur'])
83          largeur_porte_pourcentage = porte['largeur'] / mur.avoirLongueur()
84
85          pourcentage_sommet1 = porte['position'] - largeur_porte_pourcentage / 2
86          pourcentage_sommet2 = porte['position'] + largeur_porte_pourcentage / 2
87          sommet1 = mur.avoirPositionPourcentage(pourcentage_sommet1)
88          sommet2 = mur.avoirPositionPourcentage(pourcentage_sommet2)
89          return sommet1, sommet2
90
91      def ajouterMurEtPortes(self, sommets):
92          #Le tri étant lexicographique selon (x, y) et les sommets étant
93          #soit à x constant soit à y constant on fait un .sort() pour
94          #avoir leurs position sur le mur
95          sommets.sort()
96          for k in range (0, len(sommets) - 1, 2):
97              self.ajouterObstacle(OsbtacleSegment(
```

```
                point1=sommets[k],
                point2=sommets[k + 1]))

    def ajouterLieuFerme(self, lieu_ferme):
        self.lieu_ferme = lieu_ferme

        sommets = {
            'gauche' :list(self.lieu_ferme.genererSommetsCote('gauche')),
            'droite' :list(self.lieu_ferme.genererSommetsCote('droite')),
            'bas' :list(self.lieu_ferme.genererSommetsCote('bas')),
            'haut' :list(self.lieu_ferme.genererSommetsCote('haut')),
        }


        for porte in self.lieu_ferme.liste_portes :

            sommet1, sommet2 = self.recupererSommetsPorte(porte)

            sommets[porte['mur']].append(sommet1)
            sommets[porte['mur']].append(sommet2)

        self.ajouterMurEtPortes(sommets['bas'])
        self.ajouterMurEtPortes(sommets['haut'])
        self.ajouterMurEtPortes(sommets['gauche'])
        self.ajouterMurEtPortes(sommets['droite'])
```

A.4 personne.py

```
from representation_categories import RepresentationCategorie
from representation import CercleDynamique
import test_point_suivre
from fonctions_annexes import convertirMetresPixels, convertirSurfacePixelsSurfaceMetres
import math
import pymunk
import math
from pymunk.vec2d import Vec2d

class Personne(CercleDynamique):

    VITESSE_MAXIMALE = convertirMetresPixels(1.3)
    COEFFICIENT_EVITEMENT = 0.4

    #On choisi la distance maximale parcouru par l'agent en une seconde
    #comme rayon de proximité
    RAYON_DE_PROXIMITE = VITESSE_MAXIMALE

```

```
19      TEST_DIRECTION = test_point_suivre.TestDichotomieCompactageObstacle
20
21      def __init__(self,
22              masse_surfacique,
23              rayon,
24              position,
25              espace):
26
27          super().__init__(
28              masse_surfacique=masse_surfacique,
29              rayon=rayon,
30              position=position)
31
32          self.force_deplacement = self.rayon * 10**4
33          self.filter = pymunk.ShapeFilter(
34              categories=RepresentationCategorie.PERSONNE.value)
35          self.espace = espace
36          self.test_direction = Personne.TEST_DIRECTION(
37              position=position,
38              espace=espace,
39              rayon=self.rayon,
40              position_voulue=self.sortieLaPlusProche())
41          self.vitesse_maximale_propre = Personne.VITESSE_MAXIMALE
42
43      def sortieLaPlusProche(self):
44          liste_centres = self.espace.lieu_ferme.avoirCentrePortes()
45          distmin = self.position.get_distance(liste_centres[0])
46          centre_min = liste_centres[0]
47
48          for centre in (liste_centres):
49              dist = self.position.get_distance(centre)
50              if dist < distmin :
51                  distmin, centre_min = dist, centre
52
53          return centre_min
54
55      def pointEstAInterieur(self, point):
56          return point.get_distance(self.body.position) < self.rayon
57
58      def personneEstTropProche(self, personne):
59          return (personne.body.position.get_distance(self.body.position)
60              < (2 + Personne.COEFFICIENT_EVITEMENT) * self.rayon)
61
62      def estTropProcheDePersonne(self):
63          return any(map(lambda personne: self.personneEstTropProche(personne),
64              self.espace.ensemble_personnes))
```

```
65
66      def estSortie(self):
67          return self.espace.lieu_ferme.pointEstAExterieur(self.position)
68
69      def avoirCarreProximite(self):
70
71          gauche = self.position.x - Personne.RAYON_DE_PROXIMITE
72          droite = self.position.x + Personne.RAYON_DE_PROXIMITE
73          haut = self.position.y + Personne.RAYON_DE_PROXIMITE
74          bas = self.position.y - Personne.RAYON_DE_PROXIMITE
75
76          return pymunk.BB(gauche,bas,droite,haut)
77
78
79      def avoirNombreDePersonnesAProximite(self):
80          personnes_proches = self.espace.bb_query(
81              self.avoirCarreProximite(),
82              pymunk.ShapeFilter(mask=RepresentationCategorie.PERSONNE.value))
83
84          return len(personnes_proches)
85
86      def avoirSurfaceZoneDeProximite(self):
87          return self.avoirCarreProximite().area()
88
89      def mettreAJourDensite(self):
90          #Densite en personnes par metres carrès
91          surface_proximite = convertirSurfacePixelsSurfaceMetres(
92              self.avoirSurfaceZoneDeProximite())
93          self.densite = (self.avoirNombreDePersonnesAProximite()
94              / surface_proximite)
95
96      def miseAJourVitesseMax(self):
97          if self.densite == 0 :
98              self.vitesse_maximale_propre =  Personne.VITESSE_MAXIMALE
99          else :
100             self.vitesse_maximale_propre = (
101                 Personne.VITESSE_MAXIMALE * min(1, self.densite**(-0.8)))
102
103     def traiterVitesse(self):
104         if self.corps.velocity.length > self.vitesse_maximale_propre :
105             self.corps.velocity.length = self.vitesse_maximale_propre
106
107     def mettreAJourForce(self):
108         if not self.estSortie():
109             self.test_direction.update(self.position)
110             force = self.test_direction.point_a_suivre - self.body.position
```

```
111             if force != Vec2d(0, 0):
112                 force.length = self.force_deplacement
113             self.body.apply_force_at_local_point(force, Vec2d(0, 0))
114 
115     def update(self):
116         self.mettreAJourDensite()
117         self.miseAJourVitesseMax()
118         self.traiterVitesse()
119         self.mettreAJourForce()
```

A.5 test_point_suivre.py (Modélisation du mouvement)

```
1  import pymunk
2  from representation import Rectangle
3  from fonctions_annexes import convertirMetresPixels
4  import math
5  import functools
6  import geometrie
7  import collections
8  import space_hash
9  from pymunk import Vec2d
10 import base
11 import itertools
12 
13 
14 class TestBase(object):
15     '''Keyword arguments: espace, position, rayon, position_voulue
16 
17         Toute sous classes doit redéfinir la fonction `update` et
18         appeler `fin_update` à la fin de la mise à jour
19     '''
20 
21     def __init__(self, **kwargs):
22         self.rappelle_update = lambda test: None
23 
24         self.espace = kwargs['espace']
25         del kwargs['espace']
26 
27         self.rayon = kwargs['rayon']
28         del kwargs['rayon']
29 
30         self.position_voulue = kwargs['position_voulue']
31         del kwargs['position_voulue']
32 
33         self.position = kwargs['position']
34         del kwargs['position']
```

```
35
36          self.point_a_suivre = self.position_voulue
37
38          super().__init__(**kwargs)
39
40      def update(self, position):
41          self.position = position
42
43      def fin_update(self):
44          self.rappelle_update(self)
45
46  class TestGradient(TestBase):
47      #Ce test n'utilise pas la position_voulue
48
49      treillis_interet = dict()
50
51      @property
52      def treilli_interet(self):
53          return TestGradient.treillis_interet[self.espace]
54
55      def avoirDirectionASuivre(self):
56          return TestGradient.treillis_interet[self.espace].avoirGradiantPosition(
57              self.position)
58
59      def update(self, position):
60          super().update(position)
61
62          direction = self.avoirDirectionASuivre()
63
64          #Si la direction est nul la personne s'est retrouvé dans un obstacle
65          #On lui donne une direction arbitraire pour le sortire
66          if direction == Vec2d(0, 0):
67              direction = Vec2d(0, 1)
68
69          direction.length = 25
70
71          self.point_a_suivre = direction + self.position
72
73          self.fin_update()
74
75  class TestParcoursLargeur(TestBase):
76      '''Keywords Arguments: precision, valeur_defaut, cls_tableau, rayon,
77              position, position_voulue, espace
78
79          Permet de faciliter l'utilisation d'un parcours en largeur sur des
80          QuadrillageEspace
```

```
81
82          toute sous classe doivent definire
83              `genererCaseAdjacentes`
84              `genererDebutsEtValeurs`
85              `assignerValeurCase`
86      '''
87
88      def __init__(self, **kwargs):
89          self.precision = kwargs['precision']
90          del kwargs['precision']
91
92          self.valeur_defaut = kwargs['valeur_defaut']
93          del kwargs['valeur_defaut']
94
95          self.cls_tableau = kwargs['cls_tableau']
96          del kwargs['cls_tableau']
97
98          super().__init__(**kwargs)
99
100     def initialiserTableau(self):
101         tableau = self.cls_tableau(
102             position=self.espace.lieu_ferme.position
103                 - 2 * Vec2d(self.precision, self.precision),
104             hauteur=self.espace.lieu_ferme.hauteur + 4 * self.precision,
105             largeur=self.espace.lieu_ferme.largeur + 4 * self.precision,
106             precision=self.precision,
107             valeur_defaut=self.valeur_defaut)
108
109         base.parcoursEnLargeur(
110             self.genererDebutsEtValeurs(tableau),
111             self.genererCaseAdjacentesParcoursLargeur,
112             self.assignerValeurCase,
113             tableau)
114
115         return tableau
116
117     def caseEstAccessible(self, case_depart, case, tableau):
118         info_lancer_rayon = self.espace.avoirInfoSurLancerRayon(
119             tableau.avoirCentreCase(case_depart),
120             tableau.avoirCentreCase(case))
121
122         return info_lancer_rayon is None
123
124     def genererCaseAdjacentesParcoursLargeur(self, case, tableau):
125         '''Generes les cases adjacentes en prenant en comptes les obstacles'''
126         for case_adjacentes in self.genererCaseAdjacentes(case):
```

```
127             if self.caseEstAccessible(case, case_adjacentes, tableau):
128                 yield case_adjacentes
129 
130     def genererCaseAdjacentes(self, case):
131         '''Generes les cases adjacentes sans prendre compte des obstacles'''
132         raise NotImplementedError()
133 
134     def assignerValeurCase(self, case_voisine, case_courante, tableau):
135         raise NotImplementedError()
136 
137     def genererDebutsEtValeurs(self, tableau):
138         raise NotImplementedError()
139 
140 class TestGradientObstacle(TestGradient):
141 
142     DISTANCE_CHARACTERISITQUE = convertirMetresPixels(0.05)
143     DISTANCE_MAX_INFLUENCE = 3 * DISTANCE_CHARACTERISITQUE
144 
145     def transformetChampParRapportObstacle(self, tableau):
146         valeur_characteristique = min(tableau.genererValeurs())
147 
148         for case in tableau.genererCases():
149             info_point = self.espace.avoirInfoPoint(
150                 tableau.avoirCentreCase(case),
151                 TestGradientObstacle.DISTANCE_MAX_INFLUENCE)
152             if info_point is None:
153                 continue
154             valeur = valeur_characteristique * math.exp(-info_point.distance
155                 / TestGradientObstacle.DISTANCE_CHARACTERISITQUE)
156 
157             tableau[case] += valeur
158 
159 class TestGradientLargeur(TestGradient, TestParcoursLargeur):
160 
161     PRECISION_CHAMP = convertirMetresPixels(0.05)
162     INACCESSIBLE_VALEUR = -5e2
163 
164     def __init__(self, **kwargs):
165         kwargs['precision'] = TestGradientLargeur.PRECISION_CHAMP
166         kwargs['valeur_defaut'] = TestGradientLargeur.INACCESSIBLE_VALEUR
167         kwargs['cls_tableau'] = space_hash.InterpolationChampScalaire
168 
169         super().__init__(**kwargs)
170 
171         self.initialiserTreillisInteretSiNecessaire()
172 
```

```
173     def genererDebutsEtValeurs(self, tableau):
174         return zip(
175             map(
176                 tableau.avoirCasePlusProche,
177                 self.espace.lieu_ferme.avoirCentrePortes()),
178             itertools.cycle([0]))
179 
180     def genererCaseAdjacentes(self, case):
181         raise NotImplementedError()
182 
183     def assignerValeurCase(self, case_voisine, case_courante, tableau):
184         tableau[case_voisine] = tableau[case_courante] - 1
185 
186     def initialiserTreillisInteretSiNecessaire(self):
187         if self.espace not in TestGradientLargeur.treillis_interet:
188             self.initialiserTreillisInteret()
189 
190     def initialiserTreillisInteret(self):
191         TestGradientLargeur.treillis_interet[self.espace] = self.initialiserTableau()
192 
193 
194 class TestGradientLargeurObstacle(TestGradientLargeur, TestGradientObstacle):
195 
196     def initialiserTreillisInteret(self):
197         super().initialiserTreillisInteret()
198         self.transformetChampParRapportObstacle(self.treilli_interet)
199 
200 class TestChampVecteur(TestParcoursLargeur):
201     '''Choisit le mouvement des agents après avoir créer un champ
202         de vecteur vers la position voulue
203 
204         Toute sous classe doivent redéfinir la fonction `avoirCaseAdjacentes`
205     '''
206 
207     champs = dict()
208     PRECISION_CHAMP = convertirMetresPixels(0.2)
209 
210     def __init__(self, **kwargs):
211         kwargs['precision'] = TestChampVecteur.PRECISION_CHAMP
212         kwargs['valeur_defaut'] = Vec2d(1, 0)
213         kwargs['cls_tableau'] = Champ
214 
215         super().__init__(**kwargs)
216 
217         self.initialiserChampsSiNecessaire()
218 
```

```
219     def update(self, position):
220         super().update(position)
221 
222         direction = self.champ.avoirValeurPlusProche(self.position)
223         self.point_a_suivre = direction + self.position
224 
225         self.fin_update()
226 
227     @property
228     def champ(self):
229         return TestChampVecteur.champs[self.espace]
230 
231     def initialiserChampsSiNecessaire(self):
232         if self.espace not in TestChampVecteur.champs:
233             TestChampVecteur.champs[self.espace] = self.initialiserTableau()
234 
235     def genererCaseAdjacentes(self, case):
236         raise NotImplementedError()
237 
238     def assignerValeurCase(self, case_voisine, case_courante, tableau):
239         return tableau.dirigerVecteurVers(
240             case_voisine,
241             tableau.avoirCentreCase(case_courante))
242 
243     def genererDebutsEtValeurs(self, tableau):
244         for centre_sortie in self.espace.lieu_ferme.avoirCentrePortes():
245             case_sortie = tableau.avoirCaseAvecCentrePlusProche(centre_sortie)
246             vecteur = centre_sortie - tableau.avoirCentreCase(case_sortie)
247             yield case_sortie, vecteur
248 
249 
250 class TestLargeurQuatreDirections(TestParcoursLargeur):
251 
252     def genererCaseAdjacentes(self, case):
253         return case.genererCaseAdjacentes(base.Case.genererQuatreDirections())
254 
255 
256 class TestLargeurHuitDirections(TestParcoursLargeur):
257 
258     def genererCaseAdjacentes(self, case):
259         return case.genererCaseAdjacentes(base.Case.genererHuitDirections())
260 
261 class TestChampVecteurQuatreDirections(
262         TestLargeurQuatreDirections,
263         TestChampVecteur):
264     pass
```

```
265
266 class TestChampVecteurHuitDirections(
267         TestLargeurHuitDirections,
268         TestChampVecteur):
269     pass
270
271 class TestGradientLargeurQuatreDirections(
272         TestLargeurQuatreDirections,
273         TestGradientLargeur):
274     pass
275
276 class TestGradientLargeurHuitDirections(
277         TestLargeurHuitDirections,
278         TestGradientLargeur):
279     pass
280
281 class TestGradientLargeurObstacleQuatreDirections(
282         TestLargeurQuatreDirections,
283         TestGradientLargeurObstacle):
284     pass
285
286 class TestLanceRayon(TestBase):
287     '''Keywords Arguments: position, rayon, position_voulue, espace
288
289         Aide pour la contruction de test essayant d'éviter un obstacle
290         bloquant l'accès à la sortie.
291
292         Dans le code quelque chose est acceptable si aller dans sa direction
293         permer d'éviter l'obstacle bloquant.
294
295         Toute sous classe doit redéfinir les fonction `est<...>Acceptable` selon
296         ses besoins, généralement seulement `estObjetAcceptable` doit être
297         redéfinie
298     '''
299
300     def update(self, position):
301         super().update(position)
302         self.longueur_rayon = self.espace.lieu_ferme.avoirLongueurDiagonale() / 2
303         self.ininitialiserObstacleBloquant()
304
305     def avoirPositionVersAngle(self, angle):
306         direction = (self.position_voulue - self.position)
307         direction.length = self.longueur_rayon
308         direction.rotate(angle)
309         return direction + self.position
310
```

```
311         def avoirLancerAvecAngle(self, angle):
312             point_vers_lequel_lancer = self.avoirPositionVersAngle(angle)
313             return self.espace.avoirInfoSurLancerRayon(
314                 self.position,
315                 point_vers_lequel_lancer)
316 
317         def avoirObjetToucheParRayon(self, info_lancer_rayon):
318             if info_lancer_rayon is None:
319                 return None
320             return info_lancer_rayon.shape
321 
322         def avoirPointImpactRayon(self, info_lancer_rayon):
323             if info_lancer_rayon is None:
324                 return None
325             return info_lancer_rayon.point
326 
327         def avoirPointImpactVersAngle(self, angle):
328             return self.avoirPointImpactRayon(self.avoirLancerAvecAngle(0))
329 
330         def avoirObjetVersAngle(self, angle):
331             return self.avoirObjetToucheParRayon(self.avoirLancerAvecAngle(angle))
332 
333         def ininitialiserObstacleBloquant(self):
334             self.obstacle_bloquant = self.avoirObjetVersAngle(0)
335 
336         def estAngleAcceptable(self, angle):
337             objet_vers_angle = self.avoirObjetVersAngle(angle)
338             return self.estObjetAcceptable(objet_vers_angle)
339 
340         def estObjetAcceptable(self, objet):
341             return objet is None or objet is not self.obstacle_bloquant
342 
343         def estRayonAcceptable(self, info_lancer_rayon):
344             return self.estObjetAcceptable(
345                 self.avoirObjetToucheParRayon(info_lancer_rayon))
346 
347         def angleEstPlusProcheDeSortie(self, angle, autre_angle):
348             return (geometrie.avoirDistanceAngulaire(angle, 0)
349                 < geometrie.avoirDistanceAngulaire(autre_angle, 0))
350 
351         def avoirMeilleureAngleEntre(self, angle1, angle2):
352             if self.angleEstPlusProcheDeSortie(angle1, angle2):
353                 return angle1
354             return angle2
355 
356 
```

```
357 class Champ(space_hash.SpaceHash):
358     '''Keywords argument: precision, position, hauteur, largeur'''
359 
360     def __init__(self, **kwargs):
361         kwargs['valeur_defaut'] = Vec2d(1, 0)
362         super().__init__(**kwargs)
363 
364     def dirigerVecteurVers(self, case, point):
365         if self[case] == Vec2d(0, 0):
366             return
367 
368         centre = self.avoirCentreCase(case)
369         longueur_actuelle = self[case].length
370         self[case] = point - centre
371         if self[case] == Vec2d(0, 0):
372             return
373         self[case].length = longueur_actuelle
374 
375 class TestLineaire(TestLanceRayon):
376     '''Keyword Arguments: position, rayon, position_voulue, espace
377 
378         Les sous classes ne doivent pas redéfinir `update` sinon
379         elles doivent appeler `TestLanceRayon.update(self, position)`
380         au début de leur `update` au lieu de `super().update(position)`
381     '''
382 
383     PRECISION = math.pi / 100
384 
385     def update(self, position):
386         super().update(position)
387 
388         meilleur_angle = math.pi
389 
390         for i in range(0, self.avoirNombreRayon()):
391             angle_courant = i * TestLineaire.PRECISION
392             if self.angleEstPlusProcheDeSortie(meilleur_angle, angle_courant):
393                 continue
394             if self.estAngleAcceptable(angle_courant):
395                 meilleur_angle = angle_courant
396 
397         if meileur_point_a_suivre is None:
398             meileur_point_a_suivre = self.position_voulue
399 
400         self.point_a_suivre = self.avoirPositionVersAngle(meilleur_angle)
401 
402         self.fin_update()
```

```
403
404     def avoirNombreRayon(self):
405         return math.floor((2 * math.pi) / TestLineaire.PRECISION)
406
407 class TestRetiensObjet(TestLanceRayon):
408     '''Permet le retient des objet présent dans la direction
409         d'un angle pendant le temps d'une update,
410         à utiliser pour les tests devant accéder plusieurs fois à cette
411         information
412     '''
413
414     def update(self, position):
415         self.ininitialiserObjetVersAngle()
416         super().update(position)
417
418     def ininitialiserObjetVersAngle(self):
419         self.objet_vers_angle = dict()
420
421     def avoirObjetVersAngle(self, angle):
422         if angle not in self.objet_vers_angle:
423             info_lancer_rayon = self.avoirLancerAvecAngle(angle)
424             objet = self.avoirObjetToucheParRayon(info_lancer_rayon)
425             self.objet_vers_angle[angle] = objet
426         return self.objet_vers_angle[angle]
427
428
429 class TestDichotomie(TestRetiensObjet):
430
431     PRECISION = math.pi / 100
432
433     def update(self, position):
434         super().update(position)
435
436         if self.obstacle_bloquant is None:
437             self.point_a_suivre = self.position_voulue
438             self.fin_update()
439             return
440
441         #Il faut ajouter un décalage avec précision pour être sur que le
442         #milieu se trouvera au bon endroit dans la dichotomie
443         meilleur_angle = self.avoirMeilleureAngleEntre(
444             self.avoirMeilleureAngleDansIntervalle(
445                 0, math.pi - TestDichotomie.PRECISION),
446             self.avoirMeilleureAngleDansIntervalle(
447                 0, math.pi + TestDichotomie.PRECISION))
448
```

```
449             self.point_a_suivre = self.avoirPositionVersAngle(meilleur_angle)
450
451             self.fin_update()
452
453         def avoirMeilleureAngleDansIntervalle(self, angle1, angle2):
454             if (geometrie.avoirDistanceAngulaire(angle1, angle2)
455                     < TestDichotomie.PRECISION):
456                 return angle1
457
458             if geometrie.avoirDistanceAngulaire(angle1, angle2) > math.pi:
459                 raise ValueError('''Une dichotomie doit se faire sur des angles
460                     proche d'au moint pi''')
461
462             if (self.estAngleAcceptable(angle1)
463                     and self.estAngleAcceptable(angle2)):
464                 raise RuntimeError('Il n\'est pas possible d\'avoir des objet '
465                     + 'acceptable sur les deux borne de la dichotomie')
466
467             milieu = geometrie.avoirMilieuProche(angle1, angle2)
468
469             if (not self.estAngleAcceptable(angle1)
470                     and not self.estAngleAcceptable(angle2)):
471                 return self.avoirMeilleureAngleEntre(
472                     self.avoirMeilleureAngleDansIntervalle(angle1, milieu),
473                     self.avoirMeilleureAngleDansIntervalle(milieu, angle2))
474
475             if self.estAngleAcceptable(angle1):
476                 if self.estAngleAcceptable(milieu):
477                     return self.avoirMeilleureAngleDansIntervalle(milieu, angle2)
478                 else:
479                     return self.avoirMeilleureAngleDansIntervalle(angle1, milieu)
480             else:
481                 if self.estAngleAcceptable(milieu):
482                     return self.avoirMeilleureAngleDansIntervalle(milieu, angle1)
483                 else:
484                     return self.avoirMeilleureAngleDansIntervalle(angle2, milieu)
485
486
487 class TestCompactageObstacle(TestBase):
488     '''Keyword Arguments: position, rayon, espace, position_voulue
489
490         Associe à chaque obstacle les obstacles étant trop proche pour qu'un
491         disque de rayon `self.rayon` puisse passer entre les deux obstacles
492     '''
493
494     def __init__(self, **kwargs):
```

```
495         super().__init__(**kwargs)
496         self.initialiserObstacleCompacte()
497 
498     def sontConsidererMemeObstacle(self, obstacle1, obstacle2):
499         return not self.espace.peutPasserEntre(
500             self.rayon,
501             obstacle1,
502             obstacle2)
503 
504     def initialiserObstacleCompacte(self):
505         self.obstacle_compactes = dict()
506         for obstacle in self.espace.ensemble_obstacle:
507             ensemble_compacte = set()
508             for autre_obstacle in self.espace.ensemble_obstacle:
509                 if (autre_obstacle is obstacle
510                         or self.sontConsidererMemeObstacle(obstacle, autre_obstacle)):
511                     ensemble_compacte.add(autre_obstacle)
512             self.obstacle_compactes[obstacle] = frozenset(ensemble_compacte)
513 
514 
515 class TestLanceCompactageObstacle(TestLanceRayon, TestCompactageObstacle):
516 
517     def obstacleEstCompacteAvecObstacleBloquant(self, obstacle):
518         return obstacle in self.obstacle_compactes[self.obstacle_bloquant]
519 
520     def estObjetAcceptable(self, objet):
521         return (objet is None
522             or not self.obstacleEstCompacteAvecObstacleBloquant(objet))
523 
524 class TestLineaireCompactageObstacle(TestLanceCompactageObstacle, TestLineaire):
525     pass
526 
527 class TestDichotomieCompactageObstacle(TestLanceCompactageObstacle, TestDichotomie):
528     pass
529 
530 
531 class TestBordsObstacle(TestLanceRayon):
532 
533     def sommetEstAccessible(self, sommet):
534         return not self.espace.cercleEstEnDehorsDeLieuFerme(sommet, self.rayon * 2)
535 
536     def update(self, position):
537         super().update(position)
538 
539         if self.obstacle_bloquant is None:
540             self.point_a_suivre = self.position_voulue
```

```
541             else:
542                 point_impact = self.avoirPointImpactVersAngle(0)
543                 avoirDistanceAPointImpact = (
544                     lambda sommet: sommet.get_distance(point_impact))
545 
546                 sommets_accessible = filter(
547                     self.sommetEstAccessible,
548                     self.obstacle_bloquant.sommets)
549 
550                 self.point_a_suivre = min(
551                     sommets_accessible,
552                     key=avoirDistanceAPointImpact,
553                     default=self.obstacle_bloquant.sommets[0])
554 
555             self.fin_update()
556 
557     class TestProximite(TestBase):
558         '''Keyword arguments: espace, position, rayon, position_voulue, nombre_point (16)'''
559 
560         COEFFICIENT_TEST = 3
561 
562         def __init__(self, **kwargs):
563             if 'nombre_point' not in kwargs:
564                 kwargs['nombre_point'] = 16
565             self.nombre_point = kwargs['nombre_point']
566             del kwargs['nombre_point']
567 
568             super().__init__(**kwargs)
569 
570             self.ensemble_point = list(
571                 self.genererEnsemblePoint(self.rayon * TestProximite.COEFFICIENT_TEST))
572             self.point_a_suivre = self.ensemble_point[0]
573 
574         def genererEnsemblePoint(self, rayon):
575             for i in range(self.nombre_point):
576                 point_local_x = math.cos(2 * math.pi * i / self.nombre_point)
577                 point_local_y = math.sin(2 * math.pi * i / self.nombre_point)
578                 point_local = rayon * Vec2d(point_local_x, point_local_y)
579                 yield self.position + point_local
580 
581         def update(self, position):
582             self.updatePosition(position)
583             if self.espace.pointEstDansObstacle(self.point_a_suivre):
584                 self.updatePointASuivre()
585 
586             self.fin_update()
```

```
587
588     def forceUpdate(self, position):
589         self.updatePosition(position)
590         self.updatePointASuivre()
591
592     def updatePosition(self, position):
593         for point in self.ensemble_point:
594             point += position - self.position
595         self.position = position
596
597     def updatePointASuivre(self):
598         self.point_a_suivre = self.avoirPointLibrePlusProcheSortie()
599
600     def genererPointsLibres(self):
601         return filter(
602             lambda point: not self.espace.pointEstDansObstacle(point),
603             self.ensemble_point)
604
605     def avoirPointLibrePlusProcheSortie(self):
606         return min(self.genererPointsLibres(),
607             key=lambda p: p.get_distance(self.position_voulue),
608             default=self.ensemble_point[0])
```

A.6 space_hash.py (Hachage de l'espace et Treillis)

```
1  import math
2  import itertools
3  import base
4  from pymunk.vec2d import Vec2d
5  import numpy as np
6
7  class QuadrillageEspace(base.TableauDeuxDimension):
8      '''Keyword Arguments: precision, hauteur, largeur, position,
9              valeur_defaut (None)
10
11         Toute sous classe doit définir `avoirCentreCase`
12     '''
13
14     def __init__(self, **kwargs):
15         self.precision = kwargs['precision']
16         del kwargs['precision']
17
18         self.position = kwargs['position']
19         del kwargs['position']
20
21         self.hauteur = kwargs['hauteur']
```

```
22          del kwargs['hauteur']
23
24          self.largeur = kwargs['largeur']
25          del kwargs['largeur']
26
27          kwargs['nombre_lignes'] = math.ceil(self.hauteur / self.precision)
28          kwargs['nombre_colonnes'] = math.ceil(self.largeur / self.precision)
29
30          super().__init__(**kwargs)
31
32      def avoirCentreCase(self, case):
33          return Vec2d(
34              self.avoirPositionColonne(case.colonne),
35              self.avoirPositionLigne(case.ligne))
36
37      def avoirPositionLigne(self, ligne):
38          raise NotImplementedError()
39
40      def avoirPositionColonne(self, colonne):
41          raise NotImplementedError()
42
43
44  class Treillis(QuadrillageEspace):
45
46      def avoirCasePlusProche(self, point):
47          return min(
48              self.genererCasesEncadrante(point),
49              key=lambda case: point.get_distance(self.avoirPositionCase(case)))
50
51      def avoirPositionCase(self, case):
52          position_relative = Vec2d(
53              case.colonne * self.precision,
54              case.ligne * self.precision)
55          return self.position + position_relative
56
57      def avoirPositionLigne(self, ligne):
58          return self.position.y + ligne * self.precision
59
60      def avoirPositionColonne(self, colonne):
61          return self.position.x + colonne * self.precision
62
63      def avoirLigneBasse(self, point):
64          return math.floor((point.y - self.position.y) / self.precision)
65
66      def avoirLigneHaute(self, point):
67          return math.ceil((point.y - self.position.y) / self.precision)
```

```
68
69      def avoirColonneGauche(self, point):
70          return math.floor((point.x - self.position.x) / self.precision)
71
72      def avoirColonneDroite(self, point):
73          return math.ceil((point.x - self.position.x) / self.precision)
74
75      def genererCasesEncadrante(self, point):
76          ligne_haute = self.avoirLigneHaute(point)
77          ligne_basse = self.avoirLigneBasse(point)
78          colonne_gauche = self.avoirColonneGauche(point)
79          colonne_droite = self.avoirColonneDroite(point)
80
81          yield base.Case(ligne_basse, colonne_gauche)
82          yield base.Case(ligne_basse, colonne_droite)
83          yield base.Case(ligne_haute, colonne_droite)
84          yield base.Case(ligne_haute, colonne_gauche)
85
86      def reglerConflitColonnes(self, colonne_gauche, colonne_droite):
87          if colonne_droite == colonne_gauche:
88              #Le point se trouve exactement sur une colonne du treillis
89              if colonne_gauche < abs(colonne_droite - self.nombre_colonnes + 1):
90                  return colonne_gauche, colonne_droite + 1
91              else:
92                  return colonne_gauche - 1, colonne_droite
93          return colonne_gauche, colonne_droite
94
95      def reglerConflitLignes(self, ligne_basse, ligne_haute):
96          if ligne_basse == ligne_haute:
97              #Le point se trouve exactement sur une ligne du treillis
98              if ligne_basse < abs(ligne_haute - self.nombre_lignes + 1):
99                  return ligne_basse , ligne_haute + 1
100             else:
101                 return ligne_basse - 1, ligne_haute
102         return ligne_basse, ligne_haute
103
104     def avoirLigneColonnesCasesVoisines(self, position):
105         ligne_basse = self.avoirLigneBasse(position)
106         ligne_haute = self.avoirLigneHaute(position)
107         colonne_gauche = self.avoirColonneGauche(position)
108         colonne_droite = self.avoirColonneDroite(position)
109
110         ligne_basse, ligne_haute = self.reglerConflitLignes(
111             ligne_basse,
112             ligne_haute)
113
```

```
114             colonne_gauche, colonne_droite = self.reglerConflitColonnes(
115                 colonne_gauche,
116                 colonne_droite)
117 
118             return ligne_basse, ligne_haute, colonne_gauche, colonne_droite
119 
120         def avoirPositionRelative(self, point):
121             ligne_basse, ligne_haute, colonne_gauche, colonne_droite = (
122                 self.avoirLigneColonnesCasesVoisines(point))
123 
124             origine_relative = Vec2d(
125                 self.avoirPositionColonne(colonne_gauche),
126                 self.avoirPositionLigne(ligne_basse))
127 
128             return (point - origine_relative) / self.precision
129 
130     class InterpolationChampScalaire(Treillis):
131 
132         def avoirGrandientParInterpolationBilineaire(self, position):
133             #On derive la formule d'interpolation bilineaire
134             ligne_basse, ligne_haute, colonne_gauche, colonne_droite = (
135                 self.avoirLigneColonnesCasesVoisines(position))
136 
137             valeur_1_1 = self[base.Case(ligne_basse, colonne_gauche)]
138             valeur_1_2 = self[base.Case(ligne_haute, colonne_gauche)]
139             valeur_2_2 = self[base.Case(ligne_haute, colonne_droite)]
140             valeur_2_1 = self[base.Case(ligne_basse, colonne_droite)]
141 
142             y_1 = self.avoirPositionLigne(ligne_basse)
143             y_2 = self.avoirPositionLigne(ligne_haute)
144             x_1 = self.avoirPositionColonne(colonne_gauche)
145             x_2 = self.avoirPositionColonne(colonne_droite)
146 
147             delta_x = x_2 - x_1
148             delta_y = y_2 - y_1
149             dx = position.x - x_1
150             dy = position.y - y_1
151 
152             delta_f_x = valeur_2_1 - valeur_1_1
153             delta_f_y = valeur_1_2 - valeur_1_1
154             delta_f_x_y = valeur_1_1 + valeur_2_2 - valeur_2_1 - valeur_1_2
155 
156             return Vec2d(
157                 delta_f_x / delta_x + delta_f_x_y * dy / (delta_x * delta_y),
158                 delta_f_y / delta_y + delta_f_x_y * dx / (delta_x * delta_y))
159 
```

```
160     def avoirDistanceRelativeCase(self, case_1, case_2):
161         return math.sqrt(
162             (case_1.ligne - case_2.ligne)** 2
163             + (case_1.colonne - case_2.colonne)**2)
164 
165     def avoirTauxDeVariationRelatif(self, case_1, case_2):
166         dl = self.avoirDistanceRelativeCase(case_2, case_1)
167         return (self[case_1] - self[case_2]) / dl
168 
169     def avoirDeriveXCaseRelative(self, case):
170         case_1 = base.Case(1, 0) + case
171         case_2 = base.Case(-1, 0) + case
172         return self.avoirTauxDeVariationRelatif(case_1, case_2)
173 
174     def avoirDeriveYCaseRelative(self, case):
175         case_1 = base.Case(0, 1) + case
176         case_2 = base.Case(0, -1) + case
177         return self.avoirTauxDeVariationRelatif(case_1, case_2)
178 
179     def avoirDeriveXYCaseRelative(self, case):
180         case_1 = base.Case(0, 1) + case
181         case_2 = base.Case(0, -1) + case
182         derive_x_1 = self.avoirDeriveXCaseRelative(case_1)
183         derive_x_2 = self.avoirDeriveXCaseRelative(case_2)
184 
185         dy = self.avoirDistanceRelativeCase(case_1, case_2)
186 
187         return (derive_x_1 - derive_x_2) / dy
188 
189     def avoirMatriceBicubic(self, position):
190         matrice_coefficients = np.matrix([
191             [ 1, 0, 0, 0 ],
192             [ 0, 0, 1, 0 ],
193             [ -3, 3, -2, -1 ],
194             [ 2, -2, 1, 1 ] ])
195 
196         ligne_basse, ligne_haute, colonne_gauche, colonne_droite = (
197             self.avoirLigneColonnesCasesVoisines(position))
198 
199         haut_gauche = base.Case(ligne_haute, colonne_gauche)
200         haut_droit = base.Case(ligne_haute, colonne_droite)
201         bas_gauche = base.Case(ligne_basse, colonne_gauche)
202         bas_droit = base.Case(ligne_basse, colonne_droite)
203 
204         template = np.matrix([
205             [ bas_gauche, haut_gauche ],
```

```
206             [ bas_droit, haut_droit ] ])
207 
208         block_haut_gauche = base.mapMatrix(self.__getitem__, template)
209         block_haut_droit = base.mapMatrix(self.avoirDeriveYCaseRelative, template)
210         block_bas_gauche = base.mapMatrix(self.avoirDeriveXCaseRelative, template)
211         block_bas_droit = base.mapMatrix(self.avoirDeriveXYCaseRelative, template)
212 
213         block_gauche = np.concatenate(
214             (block_haut_gauche, block_bas_gauche),
215             axis=0)
216 
217         block_droit = np.concatenate(
218             (block_haut_droit, block_bas_gauche),
219             axis=0)
220 
221         matrice_valeurs = np.concatenate((block_gauche, block_droit), axis=1)
222 
223         return matrice_coefficients * matrice_valeurs * matrice_coefficients.T
224 
225     def avoirLigneVandermonde(self, scalaire):
226         return np.matrix([ 1, scalaire, scalaire**2, scalaire**3 ])
227 
228     def avoirLigneVandermondeDerivee(self, scalaire):
229         return np.matrix([ 0, 1, 2 * scalaire, 3 * scalaire**2 ])
230 
231     def avoirValeurParInterpolationBicubic(self, position):
232         x, y = self.avoirPositionRelative(position)
233 
234         X = self.avoirLigneVandermonde(x)
235         Y = self.avoirLigneVandermonde(y)
236 
237         matrice_bicubic = self.avoirMatriceBicubic(position)
238 
239         return X * matrice_bicubic * Y.T
240 
241     def avoirGradientParInterpolationBicubic(self, position):
242         #On dérive la formule d'interpolation bicubic
243         x, y = self.avoirPositionRelative(position)
244 
245         X = self.avoirLigneVandermonde(x)
246         DX = self.avoirLigneVandermondeDerivee(x)
247         Y = self.avoirLigneVandermonde(y)
248         DY = self.avoirLigneVandermondeDerivee(y)
249 
250         matrice_bicubic = self.avoirMatriceBicubic(position)
251 
```

```
252             return Vec2d(DX * matrice_bicubic * Y.T, X * matrice_bicubic * DY.T)
253
254
255         def avoirGradiantPosition(self, position):
256             return self.avoirGrandientParInterpolationBilineaire(position)
257
258     class SpaceHash(QuadrillageEspace):
259         '''Keywords argument: precision, position, hauteur, largeur, valeur_defaut (None)'''
260
261         def avoirLignePoint(self, point):
262             return math.floor((point.y - self.position.y) / self.precision)
263
264         def avoirColonnePoint(self, point):
265             return math.floor((point.x - self.position.x) / self.precision)
266
267         def avoirCasePoint(self, point):
268             return base.Case(
269                 self.avoirLignePoint(point),
270                 self.avoirColonnePoint(point))
271
272         def avoirValeurPlusProche(self, point):
273             return self[self.avoirCaseAvecCentrePlusProche(point)]
274
275         def avoirPositionLigne(self, ligne):
276             return self.position.y + (ligne + 1 / 2) * self.precision
277
278         def avoirPositionColonne(self, colonne):
279             return self.position.x + (colonne + 1 / 2) * self.precision
280
281         def avoirCaseAvecCentrePlusProche(self, point):
282             case_point = self.avoirCasePoint(point)
283             cases_proches = itertools.chain(
284                 case_point.genererCaseAdjacentes(base.Case.genererQuatreDirections()),
285                 [case_point])
286             cases_proches_valides = filter(
287                 self.__contains__,
288                 cases_proches)
289
290             distance_a_point = lambda case: self.avoirCentreCase(case).get_distance(point)
291
292             return min(cases_proches_valides, key=distance_a_point)
```

A.7 ecouteur.py

```
1   class EcouteurPersonne(object):
2
```

```
    dernier_identifiant_ecouteur = -1

    def __init__(self, personne, action):
        self.initialiserIdentifiant()

        self.personne = personne
        self.action = action
        self.personne_deja_sortie = False

        self.mettreAJourPointSuiviePersonne(self.personne.test_direction)
        self.personne.test_direction.rappelle_update = (
            self.mettreAJourPointSuiviePersonne)

    def initialiserIdentifiant(self):
        self.identifiant = EcouteurPersonne.dernier_identifiant_ecouteur + 1
        EcouteurPersonne.dernier_identifiant_ecouteur += 1

    def mettreAJourPointSuiviePersonne(self, test_point_suivre):
        self.point_suivie_personne = test_point_suivre.point_a_suivre

    def ecouter(self, temps):
        if not self.personne_deja_sortie and self.personne.estSortie():
            self.personne_deja_sortie = True
            self.executerAction(temps)

    def executerAction(self, temps):
        _action = self.action
        _action(temps)
```

A.8 lieu_ferme.py

```
import geometrie

class LieuFerme(geometrie.SimpleRectangle):

    def __init__(self,liste_portes, largeur=400, hauteur=800, position=(0, 0)):
        super().__init__(position, largeur, hauteur)
        self.liste_portes = liste_portes

    def avoirCentrePorte(self, porte):
        mur = self.avoirCote(porte['mur'])
        return mur.avoirPositionPourcentage(porte['position'])

    def avoirCentrePortes(self):
        sortie = []

```

```
        for porte in self.liste_portes :
            sortie.append(self.avoirCentrePorte(porte))

        return sortie
```

A.9 obstacle.py

```
from representation_categories import RepresentationCategorie
from representation import Representation, Rectangle, Cercle, Polygon, Segment
import pymunk

class Obstacle(Representation):
    '''Keyword Arguments: position'''

    def __init__(self, **kwargs):
        kwargs['corps'] = pymunk.Body(body_type=pymunk.Body.STATIC)
        super().__init__(**kwargs)

        self.filter = pymunk.ShapeFilter(
            categories=RepresentationCategorie.OBSTACLE.value)


class OsbtacleSegment(Obstacle, Segment):
    '''Keywords Arguments: position, point1, point2'''

    def __repr__(self):
        return 'ObstacleSegment({}, {})'.format(self.point1, self.point2)



class ObstaclePolygonale(Obstacle, Polygon):
    '''Keywords Arguments: position, sommets'''
    pass


class ObstacleRectangulaire(ObstaclePolygonale, Rectangle):
    '''Keywords Arguments: position, hauteur, largeur'''

    def pointEstAInterieur(self, point):
        return ( point.x > self.position.x and point.x < self.position.x + self.largeur
            and point.y > self.position.y and point.y < self.position.y + self.hauteur)


#Cette objet n'est pas utiliser mais pourrais être à l'avenir
#Il devra être modifié dns ce cas
class ObstacleCirculaire(Obstacle, Cercle):
```

```
40      '''Keyword Arguments: position, rayon'''
41
42      def pointEstAInterieur(self, point):
43          return self.position.get_distance(point) < self.rayon
```

A.10 representation.py

```
1   import pymunk
2   from math import pi
3   from functools import partial
4   import operator
5   import geometrie
6   from pymunk.vec2d import Vec2d
7
8   class Representation(pymunk.Shape):
9       '''Doit être instancié avec
10
11      Keyword Arguments: position, corps'''
12
13      def __init__(self, **kwargs):
14          position = kwargs['position']
15          corps = kwargs['corps']
16          del kwargs['position']
17          del kwargs['corps']
18          self.corps = corps
19          self.corps.position = Vec2d(position)
20
21      def avoirCoordoneeAbsolueDepuisRelative(self, point):
22          return self.position + point
23
24      @property
25      def corps(self):
26          return self.body
27
28      @corps.setter
29      def corps(self, value):
30          self.body = value
31
32      @property
33      def position(self):
34          return self.corps.position
35
36  class RepresentationDynamique(Representation):
37      '''Keyword Arguments: position, masse, moment'''
38
39      def __init__(self, **kwargs):
```

```
40          masse = kwargs['masse']
41          moment = kwargs['moment']
42          del kwargs['masse']
43          del kwargs['moment']
44          kwargs['corps'] = pymunk.Body(masse, moment)
45          super().__init__(**kwargs)
46
47
48  class Polygon(Representation, pymunk.Poly):
49      '''Keywords Arguments: sommets, position, corps'''
50
51      def __init__(self, **kwargs):
52          #Forcé d'appeler de cette façon car la représentation doit être
53          #créé après poly pour que le corps soit initialisé correctement
54          pymunk.Poly.__init__(self, None, kwargs['sommets'])
55          del kwargs['sommets']
56          super().__init__(**kwargs)
57
58      @property
59      def sommets(self):
60          return list(map(
61              self.avoirCoordoneeAbsolueDepuisRelative,
62              self.avoirSommetsRelatif()))
63
64      def avoirSommetsRelatif(self):
65          return self.get_vertices()
66
67      def genererAretes(self):
68          for i in range(len(self.sommets) - 1):
69               yield geometrie.SimpleSegment(self.sommets[i], self.sommets[i + 1])
70          yield geometrie.SimpleSegment(self.sommets[-1], self.sommets[0])
71
72      def avoirBaryCentre(self):
73          return (1 / len(self.sommets)) * sum(self.sommets)
74
75  class Segment(Representation, pymunk.Segment):
76      '''Keywords Arguments: point1, point2, corps'''
77
78
79      def __init__(self, **kwargs):
80          pymunk.Segment.__init__(self, None, kwargs['point1'], kwargs['point2'], 0)
81          del kwargs['point1']
82          del kwargs['point2']
83          #Le corps d'un segment ne prend apparement pas en compte la position
84          #On la met donc à 0 par défault
85          kwargs['position'] = Vec2d(0, 0)
```

```
86          super().__init__(**kwargs)
87
88      @property
89      def sommets(self):
90          return [self.point1, self.point2]
91
92      def genererAretes(self):
93          yield geometrie.SimpleSegment(*self.sommets)
94
95      @property
96      def point1(self):
97          return self.a
98
99      @property
100     def point2(self):
101         return self.b
102
103
104 class Rectangle(Polygon):
105     '''Keyword Arguments: hauteur, largeur, position, corps'''
106
107     def __init__(self, **kwargs):
108         self.largeur = kwargs['largeur']
109         self.hauteur = kwargs['hauteur']
110         del kwargs['largeur']
111         del kwargs['hauteur']
112         kwargs['position'] = kwargs['position'] + self.avoirCentreRelatif()
113         kwargs['sommets'] = list(self.genererSommetsRelatifsPymunk())
114         super().__init__(**kwargs)
115
116     def genererSommetsRelatifs(self):
117         return map(partial(operator.add, self.avoirCentreRelatif()),
118             self.genererSommetsRelatifsPymunk())
119
120     def genererSommetsRelatifsPymunk(self):
121         yield Vec2d(-self.largeur / 2, -self.hauteur / 2)
122         yield Vec2d(+self.largeur / 2, -self.hauteur / 2)
123         yield Vec2d(+self.largeur / 2, +self.hauteur / 2)
124         yield Vec2d(-self.largeur / 2, +self.hauteur / 2)
125
126     def avoirCentreRelatif(self):
127         return Vec2d(self.largeur / 2, self.hauteur / 2)
128
129 class Cercle(Representation, pymunk.Circle):
130     '''Keyword aruments: position, corps, rayon'''
131
```

```
132     def __init__(self, **kwargs):
133         rayon = kwargs['rayon']
134         del kwargs['rayon']
135         #Forcé d'appeler de cette façon car la représentation doit être
136         #créé après circle pour que le corps soit initialisé correctement
137         pymunk.Circle.__init__(self, None, rayon)
138         super().__init__(**kwargs)
139 
140     @property
141     def rayon(self):
142         return self.radius
143 
144 
145 class CercleDynamique(RepresentationDynamique, Cercle):
146     '''Keyword Arguments: position, masse, rayon'''
147 
148     def __init__(self, **kwargs):
149 
150 
151         rayon = kwargs['rayon']
152         masse = 2*pi* (kwargs['masse_surfacique'])**2
153 
154         del kwargs['masse_surfacique']
155 
156         kwargs['masse'] = masse
157         kwargs['moment'] = pymunk.moment_for_circle(masse, 0, rayon)
158 
159         super().__init__(**kwargs)
```

A.11 geometrie.py

```
1  from pymunk.vec2d import Vec2d
2  import base
3  import math
4 
5  IDENTIDIANT_COTE = [ 'gauche', 'haut', 'droite', 'bas' ]
6 
7  class SimpleSegment(object):
8      #Permet de travailler avec aise sur les segments sans devoir instancier
9      #une representation qui demande plus de place
10 
11     def __init__(self, point1, point2):
12         self.point1 = point1
13         self.point2 = point2
14 
15     def avoirPositionPourcentage(self, pourcentage):
```

```
16          return self.point1 + (self.point2 - self.point1) * pourcentage
17
18      def avoirLongueur(self):
19          return (self.point2 - self.point1).length
20
21      def avoirPositionDistance(self, distance):
22          return self.avoirPositionPourcentage(distance / self.avoirLongueur())
23
24      def __repr__(self):
25          return 'SimpleSegment({}, {})'.format(self.point1, self.point2)
26
27
28  class SimpleRectangle(object):
29      #Permet de travailler avec aise sur les rectangles sans devoir instancier
30      #une representation qui demande plus de place
31
32      def __init__(self, position, largeur, hauteur):
33          self.largeur = largeur
34          self.hauteur = hauteur
35          self.position = Vec2d(position)
36
37      def avoirLongueurDiagonale(self):
38          return math.sqrt(self.largeur**2 + self.hauteur**2)
39
40      def avoirPositionScalaire(identifiant_cote):
41          if identifiant_cote == 'bas':
42              return self.position.y
43          if identifiant_cote == 'gauche':
44              return self.position.x
45          if identifiant_cote == 'droite':
46              return self.position.x + self.largeur
47          if identifiant_cote == 'haut':
48              return self.position.y + self.hauteur
49
50      @property
51      def sommets(self):
52          ajouterPosition = lambda sommet: sommet + self.position
53          return list(map(
54              ajouterPosition,
55              self.genererSommetsRelatifSensHoraireDepuisPosition()))
56
57      def genererSommetsCote(self, identifiant_cote):
58          ajouterPosition = lambda sommet: sommet + self.position
59          return map(
60              ajouterPosition,
61              self.genererSommetsCoteRelatif(identifiant_cote))
```

```
62
63      def genererSommetsCoteRelatif(self, identifiant_cote):
64          GAUCHE = 0
65          HAUT = 1
66          DROITE = 2
67          BAS = 3
68          if identifiant_cote == 'bas':
69              arrete = BAS
70          elif identifiant_cote == 'gauche':
71              arrete = GAUCHE
72          elif identifiant_cote == 'droite':
73              arrete = DROITE
74          elif identifiant_cote == 'haut':
75              arrete = HAUT
76
77          sommets = list(self.genererSommetsRelatifSensHoraireDepuisPosition())
78          return sommets[arrete], sommets[(arrete + 1) % 4]
79
80      def avoirCote(self, identifiant_cote):
81          return SimpleSegment(*self.genererSommetsCote(identifiant_cote))
82
83      def genererSommetsRelatifSensHoraireDepuisPosition(self):
84          yield Vec2d(0, 0)
85          yield Vec2d(0, self.hauteur)
86          yield Vec2d(self.largeur, self.hauteur)
87          yield Vec2d(self.largeur, 0)
88
89      def pointEstAExterieur(self, point):
90          return ( point.x > self.position.x + self.largeur or point.x < self.position.x
91              or point.y > self.position.y + self.hauteur or point.y < self.position.y)
92
93
94  class CalculateurDistanceAvecCache(object):
95      '''Sers à mettre en cache les valeurs calculées pour éviter
96          de calculer plusieurs fois la même distance
97      '''
98
99      def __init__(self):
100         self.distances = base.KeyPairDict()
101
102     def avoirDistanceEntre(self, polygon1, polygon2):
103         return self.distances.setdefault(
104             (polygon1, polygon2),
105             self.calculerDistanceEntre(polygon1, polygon2))
106
107     def calculerDistanceEntre(self, polygon1, polygon2):
```

```
        distance_min = polygon1.sommets[0].get_distance(polygon2.sommets[0])

        for sommet1 in polygon1.sommets:
            for sommet2 in polygon2.sommets:
                distance_min = min(distance_min, sommet1.get_distance(sommet2))

        for sommet in polygon1.sommets:
            for arete in polygon2.genererAretes():
                projection = avoirProjectionSurSegment(sommet, arete)
                if projection is None:
                    continue
                distance_min = min(distance_min, sommet.get_distance(projection))

        for sommet in polygon2.sommets:
            for arete in polygon1.genererAretes():
                projection = avoirProjectionSurSegment(sommet, arete)
                if projection is None:
                    continue
                distance_min = min(distance_min, sommet.get_distance(projection))

        return distance_min

def avoirDistanceAngulaire(angle, autre_angle):
    angle_centre = angle % (2 * math.pi)
    autre_angle_centre = autre_angle % (2 * math.pi)
    if angle_centre > autre_angle_centre:
        return avoirDistanceAngulaire(autre_angle, angle)
    distance = abs(angle_centre - autre_angle_centre)
    if distance > math.pi:
        autre_angle_centre -= 2 * math.pi
        distance = abs(autre_angle_centre - angle_centre)
    return distance

def avoirMilieusAngulaire(angle1, angle2):
    '''Renvoie le milieu proche suivie du milieu eloigne'''
    angle1_centre = angle1 % (2 * math.pi)
    angle2_centre = angle2 % (2 * math.pi)

    milieu1 = (angle1_centre + angle2_centre) / 2
    milieu2 = milieu1 + math.pi
    if avoirDistanceAngulaire(milieu1, angle1) < math.pi / 2:
        milieu_proche = milieu1
        milieu_eloigne = milieu2
    else:
        milieu_proche = milieu2
        milieu_eloigne = milieu1
```

```
154     return (milieu_proche, milieu_eloigne)
155 
156 def avoirMilieuEloigne(angle1, angle2):
157     return avoirMilieusAngulaire(angle1, angle2)[1]
158 
159 def avoirMilieuProche(angle1, angle2):
160     return avoirMilieusAngulaire(angle1, angle2)[0]
161 
162 def centrerPoint(point, centre):
163     return point - centre
164 
165 def avoirPositionProjection(vecteur1, vecteur2):
166     return vecteur1.dot(vecteur2) / vecteur2.get_length_sqrd()
167 
168 def avoirProjectionSurSegment(point, segment):
169     point_centre = centrerPoint(point, segment.point2)
170     direction_segment_centre = centrerPoint(segment.point1, segment.point2)
171     t = avoirPositionProjection(point_centre, direction_segment_centre)
172     if t < 0 or t > 1:
173         return None
174     return direction_segment_centre * t + segment.point2
```

A.12 base.py

```
1  import collections
2  import numpy as np
3  import functools
4  import operator
5 
6  class EnsembleRappelle(object):
7      '''Regroupe plusieurs rappelles et renvoie l'ensemble des resultats'''
8 
9      def __init__(self):
10         self.ensemble_rappelles = []
11 
12     def ajouter(self, rappele):
13         self.ensemble_rappelles.append(rappele)
14 
15     def __call__(self, *args, **kwargs):
16         resultats = []
17         for rappele in self.ensemble_rappelles:
18             resultats.append(rappele(*args, **kwargs))
19         return resultats
20 
21 class EnsembleRappelleRenvoyantCommande(EnsembleRappelle):
22 
```

```
23      AUCUN = 0x0
24
25      def __call__(self, *args, **kwargs):
26          resultats = super().__call__(*args, **kwargs)
27          return functools.reduce(
28              operator.or_,
29              resultats,
30              EnsembleRappelleRenvoyantCommande.AUCUN)
31
32  class KeyPairDict(collections.UserDict):
33      '''Une table de hachage dont les clefs sont des pairs
34
35          A utiliser surtout dans le cas d'un petit nombre
36          d'insertions et d'un grand nombre de recuperation
37          de valeurs
38      '''
39
40      #Lorsque peut d'insertions sont faites on peut s'autoriser
41      #de stocker toutes les permutations des clefs avec la valeur
42      #associée pour ainsi éviter de devoir faire deux recherches
43      #à chaque recherche d'une clefs dans la table
44
45      def transpose(self, pair):
46          element1, element2 = pair
47          return element2, element1
48
49      def __setitem__(self, key, value):
50          self.data[key] = value
51          self.data[self.transpose(key)] = value
52
53      def __delitem__(self, key):
54          del self.data[key]
55          del self.data[self.transpose(key)]
56
57  class KeyIterableDict(collections.UserDict):
58      '''Dictionaire pouvant avoir n'importe quelle iterable comme clefs
59
60          deux iterable a, b sont considérées comme égaux lorsque
61          all(u == w for u, w in zip(a, b))
62
63      '''
64
65      #tuple est le seul type iterable qui est hachable est vérifie
66      #all(u == w for u, w in zip(a, b)) => hash(a) == hash(b)
67      #pour avoir la propriété recherché on convertie toutes les clefs
68      #en tuple
```

```
69
70      def avoirTuple(self, iterable):
71          if isinstance(iterable, collections.Iterable):
72              return tuple(map(self.avoirTuple, iterable))
73          return iterable
74
75      def __contains__(self, key):
76          return self.avoirTuple(key) in self.data
77
78      def __setitem__(self, key, value):
79          self.data[self.avoirTuple(key)] = value
80
81      def __getitem__(self, key):
82          return self.data[self.avoirTuple(key)]
83
84      def __delitem__(self, key):
85          del self.data[self.avoirTuple(key)]
86
87  class EmptyListDict(collections.UserDict):
88      '''Dictionnaire associant une liste vide à tout clefs non présente'''
89
90      def __getitem__(self, key):
91          return self.data.setdefault(key, [])
92
93  def creerListeDoubleDimension(hauteur, largeur, valeur_defaut=None):
94      return [ [ valeur_defaut for _ in range(largeur) ] for _ in range(hauteur) ]
95
96  class Case(object):
97
98      @staticmethod
99      def genererQuatreDirections():
100         yield Case(1, 0)
101         yield Case(-1, 0)
102         yield Case(0, 1)
103         yield Case(0, -1)
104
105     def genererHuitDirections():
106         yield Case(1, 0)
107         yield Case(-1, 0)
108         yield Case(0, 1)
109         yield Case(0, -1)
110         yield Case(1, 1)
111         yield Case(-1, -1)
112         yield Case(-1, 1)
113         yield Case(1, -1)
114
```

```
115     def __init__(self, ligne, colonne):
116         self.ligne = ligne
117         self.colonne = colonne
118 
119     def __add__(self, other):
120         return Case(self.ligne + other.ligne, self.colonne + other.colonne)
121 
122     def genererCaseAdjacentes(self, directions):
123         for direction in directions:
124             yield direction + self
125 
126     def __repr__(self):
127         return 'Case({}, {})'.format(self.ligne, self.colonne)
128 
129 class TableauDeuxDimension(object):
130 
131     def __init__(self, **kwargs):
132         self.nombre_lignes = kwargs['nombre_lignes']
133         del kwargs['nombre_lignes']
134 
135         self.nombre_colonnes = kwargs['nombre_colonnes']
136         del kwargs['nombre_colonnes']
137 
138         if 'valeur_defaut' not in kwargs:
139             kwargs['valeur_defaut'] = None
140 
141         self.donnee = creerListeDoubleDimension(
142             self.nombre_lignes,
143             self.nombre_colonnes,
144             valeur_defaut=kwargs['valeur_defaut'])
145         del kwargs['valeur_defaut']
146 
147         super().__init__(**kwargs)
148 
149     def __getitem__(self, case):
150         return self.donnee[case.ligne][case.colonne]
151 
152     def __setitem__(self, case, valeur):
153         self.donnee[case.ligne][case.colonne] = valeur
154 
155     def __contains__(self, case):
156         return (case.ligne >= 0
157             and case.colonne >= 0
158             and case.ligne < self.nombre_lignes
159             and case.colonne < self.nombre_colonnes)
160 
```

```
161     def __repr__(self):
162         return 'TableauDeuxDimension({}, {})'.format(
163             self.nombre_lignes, self.nombre_colonnes)
164 
165     def genererValeurs(self):
166         return map(self.__getitem__, self.genererCases())
167 
168     def genererCases(self):
169         for ligne in range(self.nombre_lignes):
170             for colonne in range(self.nombre_colonnes):
171                 yield Case(ligne, colonne)
172 
173 def parcoursEnLargeur(debuts_et_valeurs, voisins, assigner_valeur, tableau_finale):
174     '''Functions Arguments:
175             voisins(case_courante, tableau_finale),
176             valeur_case(case_voisise, case_courante, tableu_finale)
177     '''
178 
179     deja_vue = TableauDeuxDimension(
180         nombre_lignes=tableau_finale.nombre_lignes,
181         nombre_colonnes=tableau_finale.nombre_colonnes,
182         valeur_defaut=False)
183 
184     queue = collections.deque()
185 
186     for debut, valeur in debuts_et_valeurs:
187         queue.append(debut)
188         deja_vue[debut] = True
189         tableau_finale[debut] = valeur
190 
191     while len(queue) > 0:
192         case_courante = queue.popleft()
193         for case_voisine in voisins(case_courante, tableau_finale):
194             if case_voisine not in tableau_finale or deja_vue[case_voisine]:
195                 continue
196             deja_vue[case_voisine] = True
197             queue.append(case_voisine)
198             assigner_valeur(case_voisine, case_courante, tableau_finale)
199 
200 def unzip(iterable):
201     lefts = []
202     rights = []
203     for left, right in iterable:
204         lefts.append(left)
205         rights.append(right)
206     return lefts, rights
```

```
207
208 def fusioner_dictionnaires(dic1, dic2):
209     sortie = {}
210     sortie.update(dic1)
211     sortie.update(dic2)
212     return sortie
213
214 def mapMatrix(function, matrix):
215     return np.matrix(list(map(function, matrix.flat))).reshape(matrix.shape)
```

A.13 fonctions_annexes.py

```
1 RAPPORT_PIXEL_METTRES = 75
2
3 def convertirMetresPixels(mesure):
4     return mesure * RAPPORT_PIXEL_METTRES
5
6 def convertirSurfacePixelsSurfaceMetres(surface):
7     return surface / RAPPORT_PIXEL_METTRES**2
```

A.14 representation_categories.py

```
1 from enum import Enum
2
3 class RepresentationCategorie(Enum):
4
5     PERSONNE = 0x1
6     OBSTACLE = 0x2
7
8 def avoirMasqueSansValeur(masque, valeur):
9     return masque ^ valeur
```