Dynamique de foules

Grégoire Uhlrich-Meunier et Charly de Marez

2016

Dans le cadre du cours de projet numérique

Encadré par :
Christophe Winisdoerffer

SCIENCES DE LA MATIÈRE

ENS

Introduction

Dans le cadre de notre projet numérique, nous avons choisi d'étudier les mouvements de foules. La problématique est simple : comment modéliser le déplacement d'une foule de la manière la plus fidèle à la réalité possible ? Ceci dans le but de simuler diverses situations, comme par exemple l'évolutions de personnes dans des bâtiments réels.

La dynamique de foule est un domaine très vaste, qui est l'objet d'étude de nombreux chercheurs. L'attrait de la communauté scientifique pour ce champ de recherche, résulte du fait qu'il se situe à la frontière de nombreuses disciplines. En effet, en effectuant une recherche sur le sujet, on peut aussi bien tomber sur des modèles mathématiques, purement informatiques, ou encore physiques. De ce fait le problème de la dynamique de foules nous a paru compliqué de prime abord et nous ne savions pas vraiment comment l'aborder. Nous avons finalement essayé de repartir de zéro, et avons implémenté deux modèles complètement différents. L'un se basant sur la minimisation d'une fonction que nous avons défini, qui travaille sur une densité continue d'individus. L'autre qui déplace des billes une par une en résolvant un système similaire à un PFD.

Nous allons donc dans ce rapport détailler les deux modèles. Tout d'abord, nous présenterons l'approche par minimisation, son fonctionnement théorique, puis ses résultats. Ensuite nous expliquerons le principe de notre seconde approche, et enfin la dernière partie sera consacrée à comparer nos deux approches et à présenter les résultats de l'approche discrète.

Table des matières

1 Dynamique de foule : une approche par minimisation

Dans un premier temps, nous allons présenter notre première approche de la dynamique de foule. En ce qui concerne cette approche, il aurait été intéressant de discuter de toutes les étapes par lesquelles nous sommes passé pour avoir une solution fonctionnelle. Cependant, nous avons préferé mettre l'accent sur les résultats et discuter de notre méthode finale.

L'idée générale est de simuler la dynamique d'une foule en utilisant un algorithme de minimisation. La définition de la fonction qui répondra à nos attentes occupera donc une place centrale dans le problème. Ensuite, la minimisation se basant sur le principe des multiplicateurs de Lagrange, nous allons également devoir définir différentes contraintes. Enfin les résultats possibles de la minimisation devront être compris dans un certain intervalle.

Ces différents choix et définitions seront l'objet de cette première partie.

1.1 Fonction à minimiser

Imaginons la situation : une foule dans un couloir d'une largeur de 3 mètres et d'une longueur de 10 mètres. L'ensemble de la foule a pour but de sortir, la sortie étant au fond du couloir. Le réflexe (humain) de chaque individu est d'atteindre le plus rapidement possible son objectif : la sortie. Cependant les individus ne veulent pas pour autant être sérrés les uns aux autres, et désirent donc maximiser leur "espace vital".

Sur la figure 1, on considère l'individu en bleu. Celui-ci a pour objectif au cours de son déplacement de minimiser la distance D qui le relie à la sortie (indiquée en vert sur la figure). Afin de ne pas être trop collé aux autres personnes l'individu maximise par le même temps le rayon de son espace vital r_{opt}.

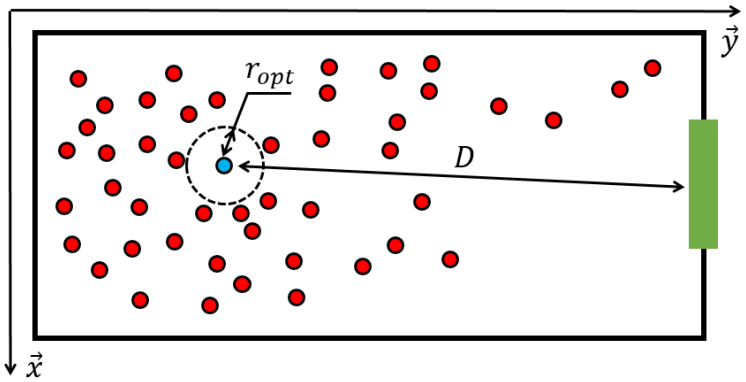

FIGURE 1: Mise en situation : une foule dans un couloir.

Plutôt que de considérer les individus comme des "pions" discernables, nous avons pris le parti de considérer la foule comme une distribution homogène. Nous avons donc définit une matrice que nous appellerons M dans la suite, ayant pour indices M_{ij} égaux au nombre de personnes en $x = i$ et en $y = j$. La discussion sur la taille des cases exprimée en mètres sera faite dans le paragraphe 1.6.

Enuite, nous avons défini une autre matrice appellée D ayant pour indices D_{ij} égaux à la distance entre la case d'indices i,j et la sortie la plus proche.. La discussion sur le calcul de cette distance sera faite dans le paragraphe 1.4.

A partir de ces deux matrices, nous définissons la fonction à minimiser comme :

$$f(M, D) = a.S_d + b.S_i \tag{1}$$

où :

$$S_d = \sum_i^n \sum_j^n D_{ij}.M_{ij} \tag{2}$$

et

$$S_i = \left(\sum_{i}^{n} \sum_{j}^{n} M_{ij}^2 \right)^{1/2} \tag{3}$$

La fonction à minimiser est ainsi une combinaison linéaire de deux termes. Le premier S_d décrit la distance des personnes à la sortie. En minimisant ce terme, la foule va tendre à s'approcher de la sortie. Le second S_i décrit l'espace vitale des individus. En effet ce terme tend à étaler la foule le plus possible car la somme de deux termes au carré est toujours plus petite que le carré de la somme : la foule se divise donc dans le plus de cases possible.

La matrice D est une constante et sera calculée avant l'éxecution de l'algorithme de minimisation. La variable est donc la matrice M. En effet à chaque incrément de minimisation, le programme devra minimiser la fonction en modifiant M ce qui signifie déplacer les individus.

La détermination des coefficients a et b ainsi que les proportions tailles de matrice/tailles des cases et donc le chiffre n seront discutés dans le paragraphe 1.6.

1.2 Les bornes

Lorsque nous allons demander à l'algorithme de minimiser notre fonction, celui-ci n'aura que l'embarras du choix pour choisir les valeurs de la matrice M qui minimiseront la fonction f. Afin d'éviter que la foule ne se téléporte à la sortie, il est primordial d'indiquer une vitesse maximale de la foule. Cela correspond dans notre cas à borner les valeurs possibles pour la matrice M. On a donc $B_{sub} \geqslant M_{ij} \geqslant B_{inf}$.

La borne inférieure est évidente, en effet le nombre de personnes peut être nul si la zone est vide d'individus, mais ne peut pas être négatif. On a donc $B_{inf} = 0$.

La borne supérieure implique de définir un certain nombre de paramètres, qui seront les paramètres intrinsèques à la foule. Le premier est le rayon de déplacement. Celui-ci a été fixé à $r_d = 1$. Le second paramètre est la densité maximale de personnes ρ_{max}, ou autrement dit, combien de personnes peut-on mettre au maximum dans une petite zone (une case de la matrice). Lorsque l'algorithme trouvera une solution pour la minimisation, celui-ci devra calculer la somme Q des cases autour de chacune des cases, plus clairement : $Q = \sum_{i=i-r_d}^{i+r_d} \sum_{j=j-r_d}^{j+r_d} M_{ij}(t)$ avec $(|i| + |j|) \leqslant r_d$.

Ensuite, le programme aura deux possibilitées : si $Q < \rho_{max}$ alors $B_{sup} = Q$, sinon $B_{sup} = \rho_{max}$.

Grace à ces bornes, l'algorithme possedera une façon de contenir la foule dans un périmètre relatif à la vitesse de déplacement voulue et au nombre maximum de personnes possibles par case.

1.3 Les contraintes

Le principe de l'algorithme de minimisation que nous allons utiliser repose sur le principe des multiplicateurs de Lagrange. Cela implique que nous allons pouvoir donner au programme des contraintes à respecter. Celui ci minimisera la fonction définie par :

$$L(M, D, t) = f(M, D) + \lambda_1 . C_1(t) + \lambda_2 . C_2(t) \tag{4}$$

La première contrainte est évidente : si personne n'est sorti du couloir, le nombre de personnes dans celui ci doit être constant. Cette contrainte permettra d'interdire à l'algorithme de supprimer des individus présent dans la matrice M.

A ce stade il est important de souligner un point important, les algorithmes de minimisations ne minimisent que des fonctions réelles, le nombre de personnes dans la matrice appartient donc à $\mathbb{R}$. Nous avons donc inclut un paramètre dans l'algorithme : ε, celui-ci octroiera une marge de manoeuvre au programme à chaque incrément de minimisation. Nous utiliserons cependant un artifice de programmation qui réajustera à chaque incrément le nombre de personnes dans l'enceinte (puisqu'on aura $\Delta M_{ij} \sim \varepsilon$).

La contrainte $C_1(t)$ s'exprime donc comme :

$$\varepsilon - \left| \sum_{i}^{n} \sum_{j}^{n} M_{ij}(t + dt) - \sum_{i}^{n} \sum_{j}^{n} M_{ij}(t) \right| \geqslant 0 \tag{5}$$

Nous avons vu plus haut, dans la partie discutant de la définition des bornes, que la foule est contenue dans un certain périmètre, ce qui garantit un déplacement de la foule à une vitesse préféfinie. Cependant rien n'empêche, à priori, les individus au sein de la foule de se déplacer d'une case à une autre contenue aussi dans le périmètre autorisé.

La contrainte C_2 a pour but de palier à ce problème que nous avons observé durant les premières simulations. En effet, on considère l'ensemble de cases de la matrice représenté sur la Figure 2, on imagine que l'ensemble de ces cases est rempli par des individus. On note n_1, n_2 et n_3 le nombre de personnes dans le périmetre délimité par les contours de couleurs, contours fixés en respect avec le paramètre r_d (égal à 1 dans cet exemple). On a ainsi $n_3 > n_2 > n_1$, et afin d'éviter que les pesonnes de la foule ne se déplace trop au sein de cette même foule, on impose que : $n_1(t) < n_2(t+dt) < n_3(t)$, ainsi les gens dans la case rouge ne peuvent pas sortir en un coup et les gens en dehors du périmètre bleu ne peuvent pas rentrer en un coup..

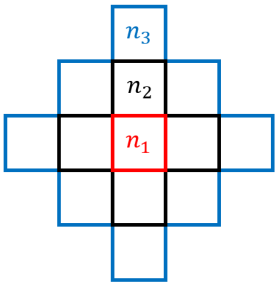

FIGURE 2

Ainsi, la contrainte C_2 est définie comme :

$$\frac{n_3(t) - n_1(t)}{2} - \left| n_2(t+dt) - \frac{n_3(t) + n_1(t)}{2} \right| \geqslant 0 \tag{6}$$

1.4 Distance à la sortie, la matrice D

Nous avons définit plus haut dans quel cadre nous allons utiliser la minimisation. En résumé nous allons minimiser une fonction $f(M, D)$ dépendant d'une variable M et d'un paramètre D. Le calcul et la manière de déterminer ce paramètre sont l'objet de ce paragraphe.

Reprenons l'exemple de la Figure 1. Si l'on place l'origine de l'axe x et de l'axe y comme indiqué sur la figure on peut considérer que la sortie se trouve en $y = 10m$. On peut donc immédiatement dire que la distance à la sortie est juste définie par la fonction $d : (x, y) \to 10 - y$. En effet, la distance à la sortie est nulle en $y = 10$ et maximale en $y = 0$, et de plus si on considère une sortie approximativement répartie sur l'ensemble du couloir, la distance ne dépend pas de x.

Si l'on considère maintenant N sorties ponctuelles (représentées par une case de la matrice), nous allons calculer pour chaque sortie la matrice D_k correspondant à la matrice représentant la distance de chaque point à la sotie k. Nous aurons ainsi une matrice en 3 dimensions de dimension $N \times n \times n$ ayant pour coefficients D_{kij} . La matrice finale en deux dimension que nous avons appelé D plus haut sera simplement calculé en prenant le minimum des k valeurs de D_{kij} à i et j fixés. On prend donc comme valeur de distance la distance à la sortie la plus proche.

L'exemple cité plus haut d'un couloir droit dépourvu d'obstacle n'est qu'un cas particulier. De plus l'intéret de la modélisation de foules est de simuler le comportement d'individus au sein d'une enceinte ou d'une géométrie plus ou moins complexe, afin d'en repérer les points sensibles par exemple.

On considère maintenant une géométrie comme celle tracé sur la Figure 3a où en rouge sont indiquées les sorties, en bleu la zone de départ de la foule et en noir les obstacles/murs infranchissables par enjambement. Il est clair que dans ce cas, déterminer une fonction d comme nous venons de le faire dans un cas simple semble beaucoup moins évident, voir impossible. En effet la matrice D doit prendre en compte le "désir" de chaque personne d'éviter l'obstacle avant même de rentrer en contact direct avec ; il faut également revoir la notion de distance à la sortie puisque le chemin n'est plus une ligne droite. Il faut donc pour cela avoir un moyen de déterminer le chemin optimal pour atteindre la sortie, et pour se faire nous avons utiliser l'algorithme A star.

Cet algorithme est un algorithme de recherche de chemin. Ce n'est pas le plus efficace, mais il est facile à mettre en place, et nous avons pu le trouver codé en python sur internet [1]. Celui ci est capable de trouver le chemin optimal entre deux points si ceux ci sont connus. Un des paramètres de cet algorithme est le nombre de cases sur lesquelles il fait la recherche de chemin autour du "point actuel" (noté dir). Dans le cadre de la minimisation nous avons fixé ce nombre à 4, cela impique que les personnes ne peuvent se déplacer qu'en ligne droite ($r_d = 1$). Dans la deuxième partie du rapport (approche discrète), nous changerons ce paramètre pour permettre aux individus de se déplacer en diagonale.

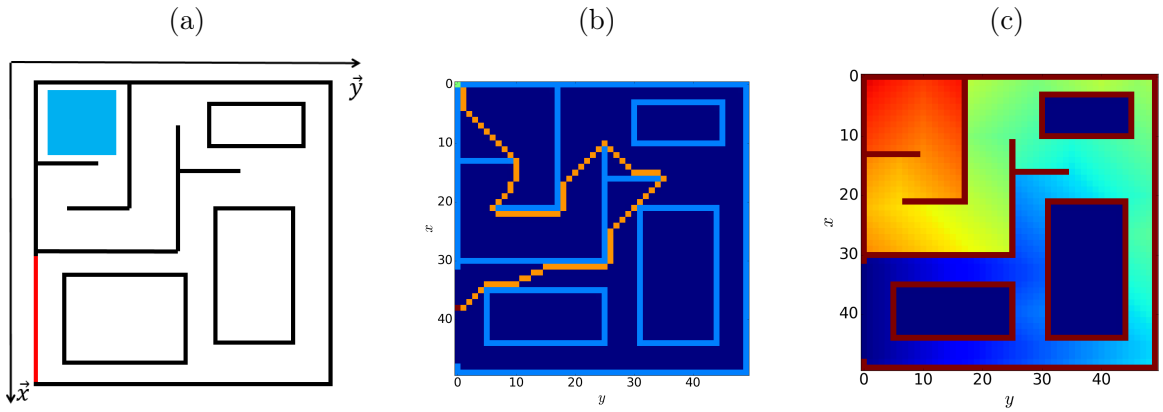

FIGURE 3: Explication du calcul de la distance.(a) Un exepmple de géométrie que l'on peut calculer avec l'algorithme A*. (b) Chemin optimal trouvé par A* entre un point quelconque de la matrice et une des sorties (ici avec dir=8 pour un meilleur aperçu). (c) Matrice distance finale obtenue avec A*.

Nous avons donc procédé comme suit : On effectue la recherche des chemins optimaux entre tous les points de la matice chaque sortie. La Figuer 3b présente le chemin calculé par l'algorithme A* entre un point et une des sorties. De cette opération on obtient une matrice de dimension $k \times n \times n$, et comme précedemment on prend le minimum sur les k possibilitées sur le premier axe de la matrice. On a donc maintenand une matrice $n \times n$ de la distance entre chaque point et une des k sorties, cette distance étant définie comme la longueur du chemin calculé par l'algorithme A*.

A la fin de ces opération, nous obtenons une matrice D qui correspond à la matrice dont nous avons parlé dans le paragraphe 1.1, qui sera utilisée comme paramètre lors de la minimisation.

Afin d'éviter que les individus ne grimpent aux murs, nous avons choisi par simplicité de remplir les cases de la matrice contenant un obstacle par la valeur n^2. En effet même en imaginant le labyrinthe le plus tordu possible, la distance à la sortie sera toujours très inférieure à cette valeur, et la minimisation ne placera donc aucun individu sur un mur.

Le résultat correspondant à la géométrie présentée dans la Figure 3a est présentée sur la Figure 3c. On retrouve les obstacles, et on observe un gradient de couleur du rouge au bleu, le bleu étant la distance minimale à la sortie et le rouge la distance maximale.

1.5 Choix du langage de programmation et des fonctions utiles

Nous avons vu dans les deux paragraphes précédents, la manière dont nous avons calculé les paramètres en jeu dans la minimisation. Afin de tester notre méthode et de la mettre en place nous avons du traduire l'aspect théorique dans un langage de programmation. Pour cela nous avons utilisé le langage python. En effet il nous a paru plus facile d'utiliser ce dernier au vu de nos connaissances immédiates sur le sujet. De plus ce langage est très avantageux du fait de la facilité et rapidité de sa mise en œuvre.

Premièrement, nous avons dû chercher une façon de réaliser la minimisation de nos fonctions. Dans la librairie Scipy, nous avons trouvé la fonction scipy.optimize.minimize qui permet de minimiser par une dizaine de méthodes différentes, avec ou sans borne, avec ou sans contrainte. Cette bibliothèque nous a permis d'avoir accès à un grand nombre de méthodes différentes, parmi lesquelles nous devions choisir la plus adaptée à notre problème. Nous avons éliminé toutes les méthodes qui ne supportaient pas de bornes et/ou de contraintes, car ne correspondant pas à notre problème. Nous avons ensuite essayé la méthode Sequential Least SQuares Programming (SLSQP) pour voir si notre approche par minimisation était réalisable ou non, sans savoir si cette méthode était la plus adaptée. La minimisation nous renvoyant des résultats prometteurs, nous avons continué dans cette direction, et il s'est avéré que la méthode de minimisation la plus efficace dans notre cas était basée sur les multiplicateurs de Lagrange,

or c'est le cas de SLSQP. Par curiosité nous avons essayé les autres méthodes de résolution de optimize.minimize qui permettent les contraintes (COBYLA, L-BFGS-B,TNC) mais aucune de ces méthodes n'a donné le moindre résultat pour notre problème. La méthode SLSQP est donc la plus efficace en théorie et en pratique pour notre problème, et c'est celle que nous avons mis en place dans nos programmes.

Dans un second temps, nous avons dû réfléchir à un moyen rapide et efficace de tracer les différentes géométries dont nous aurons besoin pour réaliser nos acquisitions. Pour ce faire, nous avons utilisé Matplotlib, qui permet d'interagir avec une figure en temps réel et ainsi de créer rapidement des formes particulières. Ce mode de création s'est avéré efficace, cependant il manque d'ergonomie lorsqu'il s'agit de tracer des géométrie complexes comme par exemple sur la Figure 3. C'est pourquoi nous avons également utilisé une autre fonction de Matplotlib : mpimg. Grâce à celle-ci, il nous suffit de dessiner avec un logiciel de traitement d'image classique (type Paint) une géométrie. Sur cette image, on indique en noir les obstacles et en rouge les sorties, et le programme s'occupe de charger cette image et de calculer la matrice D.

Enfin, pour effectuer ce calcul de distance, nous avons utilisé l'algorithme A*. Cet algorithme est très bien expliqué sur wikipedia. Nous avons préféré utiliser un code trouvé sur internet afin de gagner du temps, que nous avons trouvé ici [1].

1.6 Discussion sur les différents paramètres de la simulation

Dans notre problème, plusieurs paramètres sont déterminants pour le bon fonctionnement du programme et la représentativité de la simulation. Au cours de l'étude de notre problème, nous n'avons pas réussi à déterminer une loi générale nous permettant de calculer de façon rigoureuse les paramètres optimaux, cependant nous avons réussi à faire fonctionner de manière satisfaisante l'algorithme en choisissant des paramètres que nous allons présenter ici. Cette démarche est suffisante, puisque nous verrons dans la suite que l'approche peut être vu comme "sans échelle".

Pour le bon fonctionnement du programme, trois paramètres sont cruciaux. Le premier est la tolérance de l'algorithme de minimisation. En effet, elle ne doit pas être trop grande sinon le résultat est trop imprécis, mais elle ne doit pas non plus être trop faible car dans ce cas là la minimisation ne fonctionne tout simplement pas. Dans nos conditions standard, le choix d'une tolérance comprise entre 3 et 5 s'est avéré satisfaisant. Les deux derniers paramètres sont a et b, les coefficients présent dans l'équation (1). Là encore, phénoménologiquement, il s'est avéré que le choix d'un rapport $\frac{b}{a} = 60$ avec $a = 1$ fut satisfaisant.

Après différents tests, nous avons donc abouti à des valeurs, pour les différentes variables présentent dans les contraintes, résumés dans le tableau 1.

Taille de matrice	ρ_{max}	a	b/a	r_d	tolérance
50×50	3 personnes	1	60	1 case	3-5

Table 1: Paramètres fonctionnels pour la minimisation.

A partir de ces valeurs, et en utilisant [2] nous avons pu convertir les variables du programme en grandeurs habituelles. Dans ce document, il est dit qu'au dessus de 2 personnes présentes par mètre carré, le déplacement est possible mais ralenti. De plus nous avons estimé à 6 personnes par mètre carré la densité maximale à l'arret (en prenant en compte la corpulence moyenne d'un adulte). Ainsi, nous avons fixé à 4 personnes par metre carré la densité maximale (notée ρ). Ce document indique également une valeur de la vitesse moyenne de la marche d'un piétons comme étant égale à 1.34m/s, nous appelerons cette valeur $< v >$. Nous avons donc relié les valeurs réelles aux valeurs fonctionnant dans notre programme (cf Tableau 1), à l'aide de ces deux quantités qui résument la conversion :

— La taille d'une case de la matrice : $l_{case} = \sqrt{\frac{\rho_{max}}{\rho}}$ où ρ est la densité maximale théorique, et ρ_{max} la densité maximale que l'on fixe dans le programme.

— Le temps d'une itération : $t_{iter} = \frac{\sqrt{\frac{\rho_{max}}{\rho}}}{<v>}$

Dans notre cas, avec les valeurs du Tableau 1, on a $l_{case} = 0.86m$ et $t_{iter} = 0.65s$.

1.7 Résumé du principe de minimisation et programmes

Afin de résumer le fonctionnement de l'outil que nous avons mis en place, nous présentons ici un schéma explicatif contenant les différentes parties du programme ainsi que les données extraites de chaque.

Au final, nous possédons trois programmes. Le premier a pour fonction de créer la matrice D. Ce programme permet : soit un tracé interactif avec Matplotlib, soit le chargement d'une image au format png et la création direct de la matrice D. Ce programme renvoie donc la matrice contenant les distances dans un fichier texte, ainsi qu'un plot au format png afin de visualiser cette matrice.

Le deuxième programme est celui servant à la minimisation, il prend comme paramètre la matrice D. Dans celui ci on peut régler les différentes caracteristiques de la foule comme ρ_{max} ou la position initiale de la foule par exemple. En sortie ce programme délivre les résultats finaux : Le déplacement de la foule dans une géométrie donnée.

Nous avons également codé un programme de traitement qui permet l'étude détaillée des résultats. Les différents graphes de la partie 2 ont étés tracés à l'aide de ce programme.

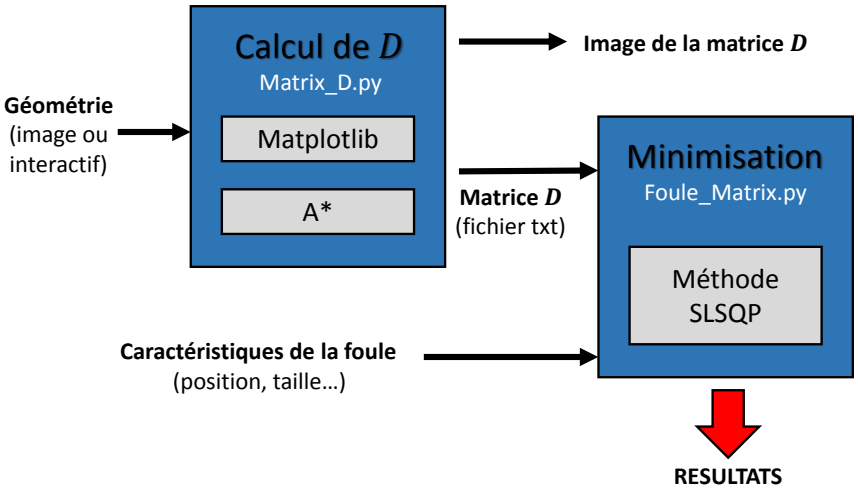

FIGURE 4: Schéma résumant le fonctionnement de notre programme

2 Déplacer une foule dans la matrice

Nous allons maintenant présenter les résultats obtenus à partir des programmes décrits dans la partie 1. Les paragraphes de cette section seront chacun consacrés à une configuration particulière dans laquelle nous développerons les différents résultats, ainsi que l'accord de ceux-ci avec la réalité. Il ne s'agit que d'un échantillon de toutes les acquisitions que nous avons faites.

2.1 Une première situation simple

Tout d'abord nous allons ici présenter notre méthode experimentale avec un exemple simple. Ici, on considère un couloir droit présentant en son centre un goulet d'étranglement. La géométrie est présenté sur la figure 5a. Cette géométrie a donc été chargée dans le programme Matrix_D.py qui a calculé la matrice D à partir de cette géométrie, le résultat est présenté sur la figure 5b. Dans ce premier exemple, la matrice est une matrice 50×50.

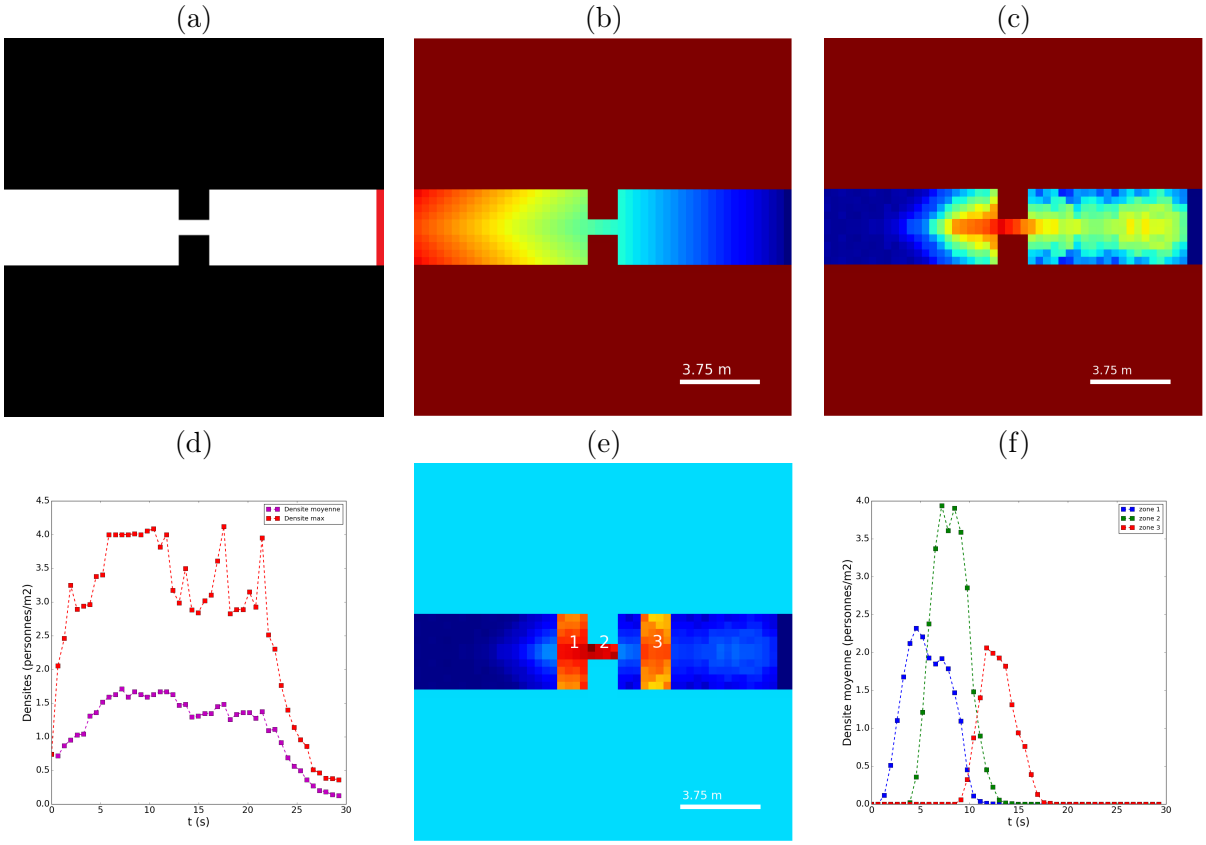

FIGURE 5: Première situation.(a) Géométrie envoyée au programme Matrix_D.py.(b) Matrice distance obtenue.(c) Carte des zones à riques.(d) Densité maximale et densité moyenne de la foule en fonction du temps.(e) Emplacement des zones étudiées plus précisémment.(f) Densité au cours du temps dans les zones étudiées.

Nous avons décidé de simuler le déplacement d'une foule de 100 personnes placées sur une surface de $25m^2$ à l'extrémité gauche du couloir. Nous éxecutons donc le programme de simulation de foule Foule_Matrix.py à partir de la matrice D obtenue plus haut. Celui ci renvoie une animation du déplacement de la foule, ainsi qu'une sauvegarde des différentes étapes du mouvement. Enfin, nous utilisons le programme de traitement, qui charge cette sauvegarde, et fournit les résultats que nous allons maintenant développer.

La première information à laquelle nous avons accès, est une "carte des zones à risques". Cette carte est une matrice obtenue en superposant toutes les itérations, puis en effectuant une moyenne temporelle. On a donc accès rapidement, aux zones critiques sur lesquelles la foule a été la plus concentrée. Le résultat pour la première situation est présenté sur la figure 5c. On voit au niveau de l'entrée du goulet un gradient de couleur. Cela indique que plus la foule s'est approchée de l'étranglement, plus celle ci s'est agglomérée afin de passer dans le petit espace.

Nous voyons également qu'après l'obstacle, la foule s'est "dissipée" pour se répartir de façon plus homogène. Ces résultats sont ceux attendus, puisque l'on sait bien qu'une foule à l'approche d'un obstacle comme celui ci va avoir tendance à ralentir, et sa concentration à augmenter. Cette exemple simple montre bien que nous avons accès à des informations pertinentes à partir de cette carte.

Ensuite, nous avons tracé la densité de la foule en fonction du temps. Ce résultat est présenté sur la figure 5d. En rouge est tracée la densité maximale, c'est à dire le maximum de la matrice M qui représente le nombre de personnes dans une case. En violet la densité moyenne de la foule. On peut voir les deux densités augmenter jusqu'à t=5s qui est le temps où les premières personnes rentrent dans le goulet. On voit ensuite la densité moyenne se stabiliser à une valeur maximale durant la traversée de l'obstacle, puis diminuer jusqu'à la sortie de toutes les personnes. On constate également que lors du passage de l'étranglement, la densité maximale atteint la densité critique (4 personnes par m^2), ce qui veut dire que les personnes au sein de la foule sont très serrées dans l'obstacle. Ensuite la densité maximale va décroître jusqu'à la sortie. Durant cette décroissance, on observe cependant quelques pics atteignant la densité maximale. Ceux ci sont dû à des erreurs commisent lors de la minimisation par le programme, ils ne faut donc pas les prendre en compte lors de l'interprétation des résultats.

Enfin, grâce au programme de traitement, nous pouvons avoir accès à la densité de la foule dans une zone souhaitée. Dans ce premier cas, nous avons considéré trois zones qui sont présentées sur la figure 5e. La première zone correspond à l'entrée du goulet, la deuxième à l'intérieur de celui ci et enfin la dernière est une zone de même taille que la zone 1 mais dans le couloir de taille normale. Le résultat est celui de la figure 5f. Avec ce graphique, nous avons accès principalement à deux informations. La première est la concentration des personnes dans une zonne donnée. On voit ici que comme prévu la concentration est bien plus importante dans la zone 2. Dans les deux autres zones, la concentration est à peu près équivalente. Ce résultat est logique puisque les valeurs de densités à chaque temps sont obtenues en moyennant sur toute la zone, et nous n'avons donc pas accès à l'entrée exacte du goulet. Cependant nous avons accès à une autre information qui nous permet de différencier ces deux cas. En effet on peut voir que le pic correspondant à la zone 1 est bien plus large que celui de la zone 3. Cela nous indique qu'avant l'obstacle, la foule a piétinnée, et a dû attendre avant de rentrer. A partir de la largeur des pics, nous pouvons donc voir que la foule s'est agglutinée dans la zone 1 alors qu'elle n'a fait que passer rapidement dans la zone 3, ce qui était un résultat attendu.

En résumé, nous avons pu voir avec un exemple simple, que notre programme fournit des résultats en accord avec le sens commun, et nous allons dans la suite le tester dans des cas plus complexes.

2.2 L'entrée dans le métro

Nous avons enusite fait fonctionner le programme avec une situation inspirée de la vie de tous les jours : le métro. Pour cela nous avons tracé la géométrie présentée sur la figure 6a. Sur celle-ci, on a placée un mur comportant des entrées qui représente la rame de métro. Nous allons placer la foule à gauche de la rame, et afin la de mettre en mouvement, nous avons mis une longue sortie sur le bord droit. La figure 6b représente la matrice distance.

Une foule de 100 personnes, initiallement répartie sur $30m^2$ a été placé à l'éxtrémité gauche de l'enceinte et s'est donc comme prévu déplacée jusqu'aux sorties. Cela simule bien l'entrée d'une foule de 100 personnes dans le métro.

Le premier résultat intéressant, que l'on peut voir sur la carte des zones à risques (figure 6c), est que la foule s'est organisée en files devant les portes. Cela provient du fait que lors du calcul de la matrice D, les distances sont effectivement plus courtes si l'on se place dans l'axe d'une porte, la minimisation privilégie donc ces zones. Cela est en accord avec les observations que l'on peut faire dans le metro, où les gens (venant de loin) marche en s'alignant avec les portes. On voit ensuite sur cette même figure que les zones sensibles sont comme prévu les portes, nous étudierons donc plus en détail dans la suite ces zones. Enfin on voit sur cette figure que les 5 entrées présentent un profil de densité identiques, cela nous conforte sur la reproductibilité de nos experiences. Nous verons également dans la suite que cela est confirmé par une étude plus fines de ces profils de densité.

Sur la figure 6d, on voit que la densité maximale fluctue fortement. Cela est dû à l'attrait des individus pour les zones alignées avec les portes, qui n'hésitent pas à se serrer pour être aux meilleurs positions afin de préparer leur arrivée aux portes. Cependant on voit bien que la densité max atteint son maximum à t=20s environ. Cet instant correspond au moment où la plupart des individus passent par la porte. De plus, la densité moyenne est maximale au moment du passage de la porte, c'est donc ce passage que nous avons observé plus en détails à travers

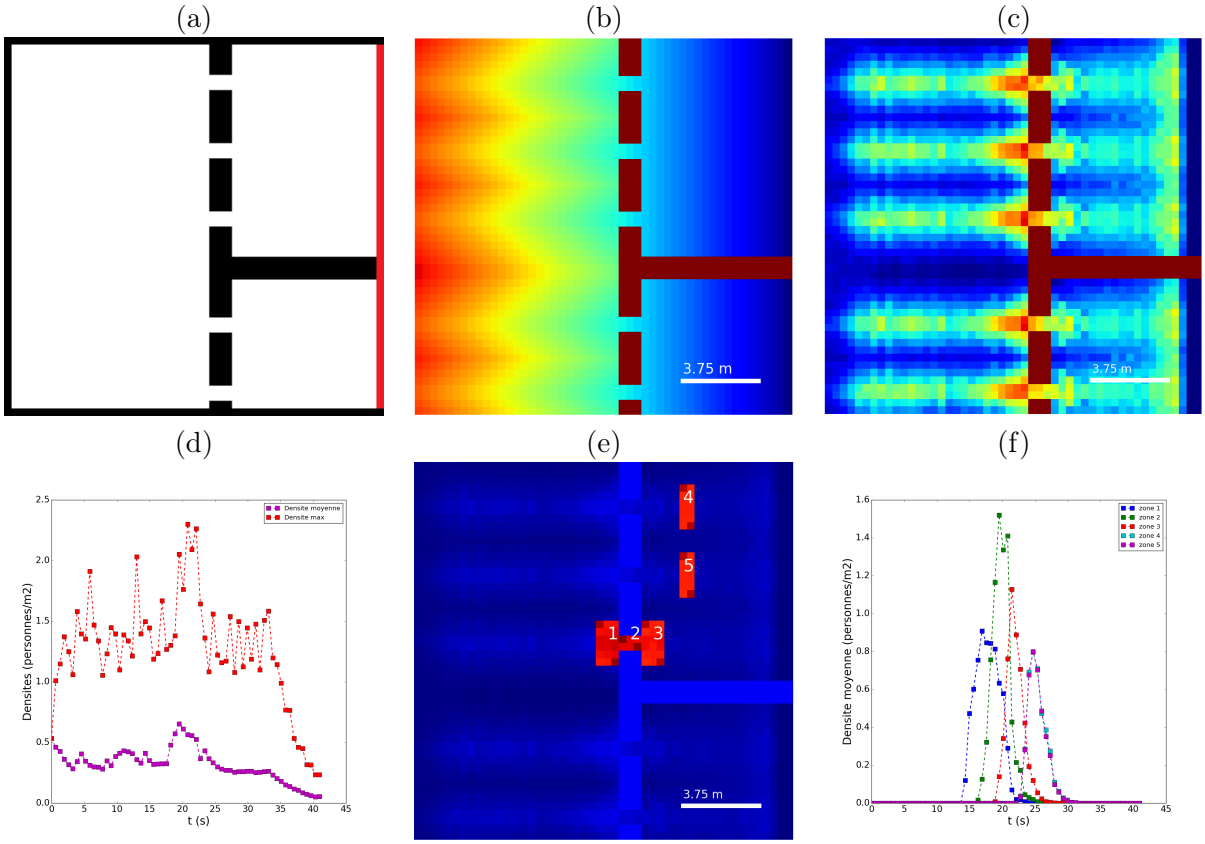

FIGURE 6: Entrée d'une foule dans le métro.(a) Géométrie envoyée au programme Matrix_D.py.(b) Matrice distance obtenue.(c) Carte des zones à riques.(d) Densité maximale et densité moyenne de la foule en fonction du temps.(e) Emplacement des zones étudiées plus précisémment.(f) Densité au cours du temps dans les zones étudiées.

les zones 1, 2 et 3 présentées sur la figure 6e. Ici l'étude des profils de densités (figure 6f) relatifs à ces zones est strictement identique à celle faite au paragraphe 2.1, nous ne la détaillerons pas plus. Nous pouvons juste dire que la présence de plusieurs portes n'a pas altéré le fonctionnement du programme. Enfin nous avons comparé les profils des zones 4 et 5, et on constate que ceux ci sont parfaitement identiques (figure 6f). Cela nous confirme bien la reproductibilité des experiences.

2.3 Une géométrie plus complexe, ou comment essayer de piéger le programme

Maintenant que nous avons vu l'efficacité (ou non, cf Annexe A) du programme sur des géométrie assez basiques, nous allons présenter dans ce paragraphe un cas où la géométrie est très complexe, et voir dans quelle mesure l'algorithme est efficace.

Nous avons premièrement conçu un labyrinthe comportant différents changements de direction ainsi que des rétrecissements de couloirs. Nous verrons par la suite qu'un paramètre est important pour la réussite de la simulation : la plus grande distance dans la matrice D (figure 7a). Ici, la plus grande distance est de 180 cases. Cela veut dire qu'une personne placé sur cette case se trouve à $180 \times 0.375 = 67.5m$ de la sortie la plus proche.

Nous avons donc pû executer (dans ce cas sans problèmes) le programme de simulation de foule avec dans un premier temps une foule de 50 personnes, puis une foule de 100 personnes. Les cartes de zones à risques relatives à ces deux acquisitions sont présentées sur les figures 7b et 7c. On peut voir sur ces cartes, les zones les plus fréquentées par la foule. Sur les deux figures, les zones les plus problématiques sont les mêmes, ce sont les couloirs les plus fin, et en particulier les coins presents dans ces couloirs. Rien d'étonnant, mais cela confirme des prédictions intuitives et des résultats connus (nous n'avons cependant pas trouvé de bibliographie serieuse à ce sujet).

On voit que les deux cartes sont, à quelques petits détails près, identiques. Cela nous dit que cette carte fournit un bon outil d'analyse qualtitative puisqu'il fournit des résultats facilement analysable et consistants avec

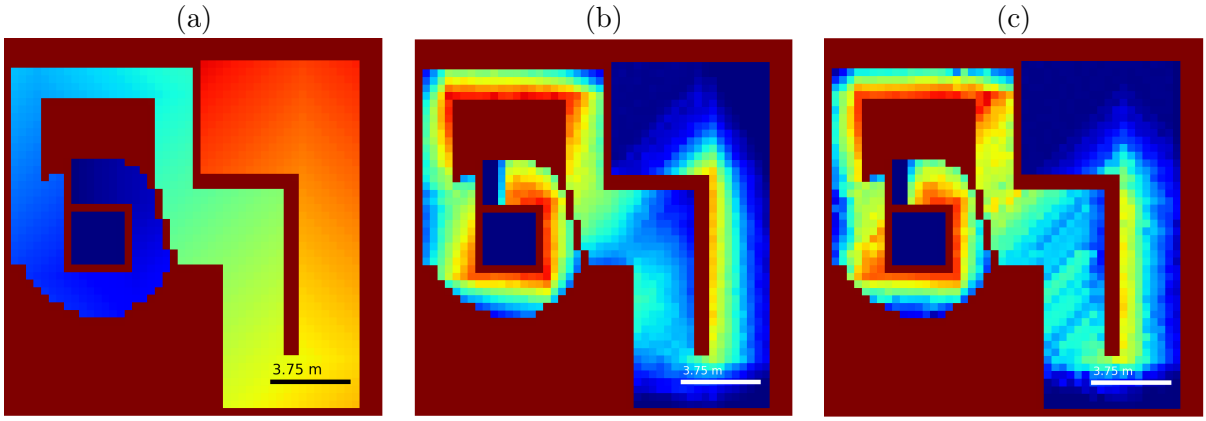

FIGURE 7: Premier labyrinthe.(a) Matrice distance correspondant au premier labyrinthe considéré.(b) Carte des zones à risques pour une simulation avec 50 personnes dans la foule.(c) Carte des zones à riques pour 100 personnes.

la prédiction, sans forcément mettre en jeu un grand nombre d'individus. Sur ces acquisitions, le programme s'est donc révélé concluant, et satisfaisant quant à nos attentes.

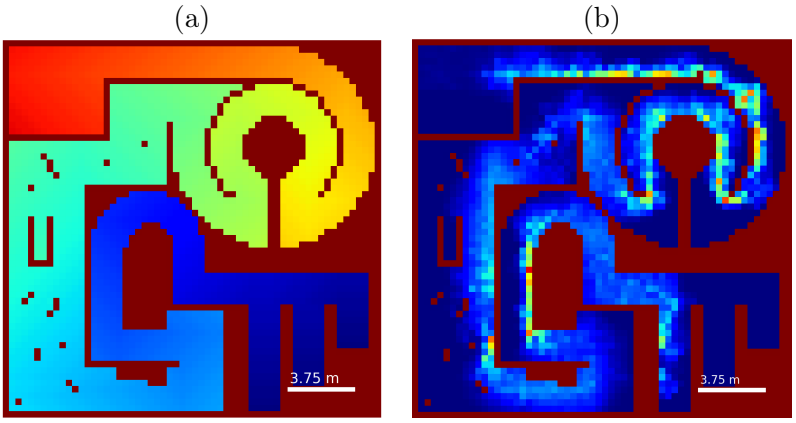

FIGURE 8: Second labyrinthe.(a) Matrice distance correspondant au second labyrinthe considéré.(b) Carte des zones à risques pour une simulation avec 10 personnes dans la foule.

Dans un second temps, nous avons accentué la difficulté. En effet nous avons conçu une géométrie sur une matrice 60×60 comportant toutes les difficultés possibles : virages, demi-tours, obstacles ponctuels, obstacles longs, plusieurs sorties, cul de sac et couloirs fins. La matrice distance relative à ce labyrinthe est présenté sur la figure 8a.

Le premier point sur lequel on peut insister est le suivant : le calcul de la matrice est concluant, on voit des gradients de couleurs cohérents avec ce à quoi on pouvait s'attendre. Le programme de calcul de la matrice D est donc une réussite. Seul petit problème, le calcul dure environ 2 heures. Cependant on peut mener différentes mesures sans avoir à la recalculer.

C'est lors de l'éxecution du programme de la foule que nous avons rencontré des problèmes. En effet, la minimisation a eu beaucoup de mal à finir la simulation sans échec. Pour la mener à bien, nous avons été obligé d'augmenter la tolérance de la fonction de minimisation jusqu'à une valeur de 10, et n'avons pû faire se déplacer qu'une foule de 10 personnes. Nous avons conclu que ce problème est dû aux valeurs présentes dans la matrice D. En effet, la plus grande valeur est égale à 312. Les valeurs des fonctions à minimiser sont donc très importante, et la fonction ne doit pas réussir à gérer des nombres si grands. On voit donc une nouvelle limite à notre approche par minimisation : les distances trop grandes à parcourir.

Partant du fait vu plus haut que la carte de zones à risques n'est pas dépendante du nombre de personnes dans

la foule, nous avons présenté sur la figure 8b la carte de zones à risques pour une simulation avec 10 personnes. Ces individus sont placés au début de la simulation dans le coin en haut à gauche. Ce placement de foule initial a l'avantage de faire balayer à la foule tout le labyrinthe. Bien entendu, les zones sont peu visitées donc les gradients de couleurs sont faibles. Cependant on voit bien apparaître des lieux plus sensibles que d'autres.Ces zones sensibles correspondent à des zones de ralentissement des individus, et non d'entassement (au vu du faible nombre d'individu dans le labyrinthe). Sur cette carte, les zones à problèmes se situent principalement sur les murs que longent les individus. Les points les plus critiques semblent être les "demis-tours" que les individus ont à faire dans la zone circulaire. Cela semble cohérent avec l'observation de personnes se mouvant dans un tel couloir : un ralentissement lorsqu'il y a un virage très sérré à faire.

On peut donc voir à travers ces deux exemples que le programme de minimisation s'est avéré efficace pour nous indiquer des zones sur lesquelles les individus de la foule ont été ralentis. Cependant, le programme, a montré des faiblesses importantes. Ces faiblesses n'ont pas pu être résolues, car elles sont intrinsèques à la manière dont nous avons abordé la dynamique de foule, c'est à dire la minimisation. Nous allons donc dans le paragraphe suivant resumer les différents points forts et points faibles de cettre première approche de la dynamique de foule.

2.4 Conclusion sur l'approche par minimisation

Nous avons vu, dans les paragraphes 2.1 et 2.2, que notre méthode de mise en mouvement de foule, s'est avéré concluante, et que nous avons pû tirer de bonnes conclusions des acquisitions. Cette approche est donc concluante si l'on considère des cas simples.

Cependant, il s'est avéré au cours d'autres acquisitions (cf Annexe A) que des résultats que nous pensions évidents n'apparaissent pas, voir que les résultas aillent à l'encontre des prédictions. Ces problèmes viennent du fait que nous controllons le déplacement de la foule, mais pas les interactions au sein de celle-ci. Il en résulte que nous ne savons pas exactement ce que la fonction de minimisation décide au sein de la foule. Nous ne pouvons donc pas voir d'effets comme l'étalement de la foule puisque le programme peut choisir (dans une certaine mesure) des déplacements que nous n'avons pas pu interdire car trop complexes.

Ensuite, nous avons rencontré des limitations de deux sortes. Premièrement celle des distances trop grandes. En effet, on a pû voir dans le paragraphe 2.3 que si les distances à parcourir sont trop grandes, les valeur des fonctions à minimiser sont trop importantes. Ainsi, le programme n'arrive pas à gérer de grands nombres, avec toutes les contraintes que nous lui imposons (obstacles, cotntraintes, bornes, vitesses..). Nous ne pouvons donc pas créer des foules de beaucoup d'individus si celles ci ont un trop gros chemin à parcourir, et nous n'avons pas trouvé un autre jeu de paramètre permettant de faire fonctionner le programme dans ces conditions. Le second problème vient du temps de calcul. En effet pour donner un ordre d'idée, la minimisation sur une zone de 40×40 prend environ 8h. Nous nous sommes arrangé dans notre programme pour que la minimisation ne se fasse que sur la zone où des gens sont présents. De cela résulte que nous ne pouvons créer que des foule contenues sur des petite surfaces. Or, nous avions prévu d'étudier des géométrie inspirés de batiments reèls comme la gare de la part dieu, cependant la taille des matrices dans ce cas aurait été problématique. En effet, pour simuler l'évacuation d'un tel batiment, nous devons placer des gens partout, et donc le temps de calcul dans ce cas aurait été beaucoup trop important pour être envisagé.

En revanche, le temps de calcul est raisonnable pour des petites foules. Il nous a donc été permis de faire se déplacer dans un couloir une petite foule. Si l'on fait cela, la foule peut balayer un labyrinthe, ou tout autre géométrie du type, et ainsi nous fournir de bonnes informations. En effet à partir d'une telle acquisition, nous avons accès à la carte de zones à risques. Celle ci nous renseigne de manière qualitative sur des endroits à problème dans une géométrie en cas de passage d'une foule. On peut très bien imaginer ainsi de reproduire à une échelle arbitraire un couloir, et à partir d'une approche par minimisation repérer les points sensibles avant de faire ensuite une approche plus fine.

En testant l'approche par minimisation avec différents paramètres (vitesse, densité max...) il s'est avéré que les cartes de zones à risques étaient sensiblement équivalentes. Nous avons donc conclu de cela que si les résultats sont cohérent à une certaine échelle arbitraire, ils seront équivalents en changeant les paramètres. Cette approche est donc " sans échelle ".

Enfin le point très positif de cette approche a été la création de la matrice distance. En effet, celle ci décrit au mieux le désir d'un individu dans un couloir, si celui ci connaît la géométrie à l'avance.

Dans les deux premières parties, nous nous sommes attachés à expliquer notre approche par minimisation de la dynamique de foule. Bien que nous ayons pû aboutir à des résultats satisfaisant, cette méthode s'est révélée pleine d'imprécisions ainsi que de faiblesses. Compte tenu du fait que nous avons entièrement imaginé cette méthode (excepté A*), nous sommes satisfait des résultats obtenus. Si l'on fait des recherches sur la dynamique de foule, la plupart des simulations reposent sur des techniques de milieux granulaires, et modélisent les individus par des billes ([2] ou [3]). C'est pourquoi dans les parties suivantes, nous allons décrire la deuxième partie de notre travail sur ce projet numérique dans laquelle nous avons étudié la dynamique de foules à l'aide d'une seconde approche.

3 Dynamique de foule : une approche discrète

Simuler le déplacement d'une foule en assimilant les individus à des billes a été la première idée que nous avons eu, bien avant la minimisation. Cependant nous avons été vite freinés par un grand nombre de contraintes. Ce qui nous a fait revenir vers cette approche est simple : la matrice distance. En effet, nous allons voir dans la suite que le déplacement des pions, billes ou individus (désignations équivalentes que nous utiliserons dans la suite) est régit par cette matrice.

Dans cette partie, nous allons expliquer le principe de cette nouvelle approche. Le programme repose sur une boucle, dans laquelle on effectue à chaque itérations une série d'actions. Nous décriront dans les paragraphes de cette partie les différentes actions effectuées à chaque itérations.

3.1 Vitesse idéale

La première étape d'une itération, consiste à faire se déplacer les billes suivant leurs chemin idéal. Ce déplacement se fait sans se préoccuper des billes autour. Pour cela nous avons utilisé la matrice distance présentée dans la première section. Il est important de noter que dans la partie minimisation, l'algorithme A* était réglé pour chercher un chemin dans les 4 cases autour de lui. Ici en revanche, rien n'empêche les billes d'explorer les 8 cases alentours, c'est pourquoi l'algorithme A* a dans ce cas été réglé pour rechercher son chemin sur 8 cases.

La vitesse idéale d'une bille placée à une position (x, y) est calculée à partir du gradient de la matrice distance. Pour se représenter le principe, on considère une matrice distance comme présentée sur la figure 9a. Si on représente en 3D cette matrice, on obtient la figure 9b. Si on pose une bille au centre, au plus haut : celle-ci chute jusqu'aux sorties. C'est à peu près le principe du déplacement idéal, en considérant une vitesse constante durant la descente (pas d'accélération, l'analogie s'arrête ici). Nous calculons donc le gradient de cette matrice afin d'obtenir un champ de vitesse, le résultat est présenté sur la figure 9c.

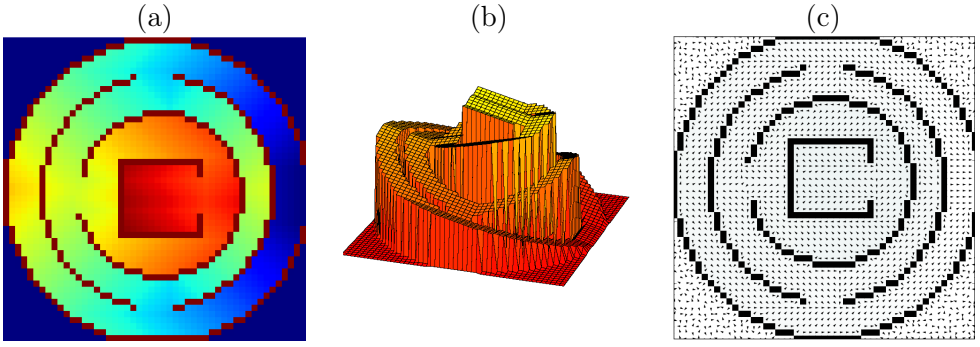

FIGURE 9: Illustration du principe de vitesse idéale.(a) Matrice distance.(b) Représentation d'une analogie avec une pente. (c) Champ de vitesse qui régit le déplacement idéal d'une bille.

Ainsi, un individu a un objectif, un chemin pour l'atteindre, et donc un module de vitesse. Sa première envie va être de suivre ce chemin avec sa vitesse de déplacement propre. Bien sûr, cela entraine des collisions entre les billes et c'est pourquoi les étapes suivantes sont là.

3.2 Gestion des interactions entre individus

Afin de simuler le contact entre les pions, nous avons choisi une force dérivant d'un potentiel harmonique du type $V = \frac{1}{2}k.x^2$. Le contact est ainsi modélisé par la partie répulsive d'une force de rappel d'un ressort.

Plus précisément, si on a deux billes de rayon r_i et r_j, la force entre les deux billes est différente de zéro si et seulement elles sont en contact :

$$\overrightarrow{F_{i\to j}} = \begin{cases} 0 & \text{si } ||\overrightarrow{d_{ij}}|| > r_i + r_j \\ k.\overrightarrow{d_{ij}} & \text{sinon.} \end{cases}$$

Et on a bien sur $\overrightarrow{F_{i\to j}} = -\overrightarrow{F_{j\to i}}$. A partir de cette force, on calcul la variation de position comme :

$$\overrightarrow{\Delta x_i} = \sum_{j=1}^{N} \overrightarrow{F_{j\to i}}.\frac{dt^2}{2} \tag{7}$$

Nous faisons ainsi un PFD dans lequel m=1, et donc forces et accélérations sont équivalentes.

Dans la partie minimisation, nous avons vu que pour simuler plus de réalisme, nous avions ajouté la notion d'espace vital entre les individus. Ici nous avons fait de même en ajoutant une force de même type que $\overrightarrow{F_{i\to j}}$, mais agissant à plus longue distance, et ayant une constante de raideur k' inférieure.

3.3 Obstacles

Pour gérer les obstacles, nous avons procédé exactement de la même façon que pour les interactions entre individus. En effet, une force de la même forme que $\overrightarrow{F_{i\to j}}$ empêche les pions de pénetrer dans les obstacles, avec une constante de raideur très forte.

3.4 Resumé du principe de la simulation

En résumé :

- Les billes se déplacent librement en suivant un chemin optimal ;
- Les billes en collisions se repoussent jusqu'à ne plus se toucher ;
- Les billes s'écartent entre elles à distance, afin de modéliser un "espace vital" ;
- Les billes s'écartent des murs pour revenir sur le droit chemin.

Pour une bonne précision de ces forces et intéractions, il faut que le pas de temps *dt* soit court. Mais pas trop, sinon les variations de positions dûes aux forces sont trop faibles pour être calculées par l'ordinateur.

Au final, nous avons les positions des billes à chaque itération. A cause de tous ces ressorts, les billes suivent le bon chemin mais oscillent autour de celui-ci. Nous avons donc décidé de moyenner les déplacements par tranches de 10 itérations pour supprimer ces oscillations, et ainsi pu observer un mouvement fluide et représentatif de la réalité.

Comme avec l'approche par miminisation, nous avons pu voir que cette approche ne dépend pas non plus de l'échelle que l'on considère. C'est pourquoi dans la suite nous nous sommes attaché à décrire de manière la plus réaliste possible une situation, sans s'attacher à quantifier les résultats. Par exemple nous n'avons pas converti les vitesses en m/s, nous avons cependant respecté les proportions entre les obstacles, le rayon des pions et la vitesse. En effet, augmenter les vitesses ne changerait pas le résultat final, si ce n'est changer la précision. Nous ne nous sommes donc pas intéressés au quantitatif, qui aurait nécessité une étude plus poussée des réels mouvements de foules, mais plutôt au qualitatif. Nous avons également fixé phénoménologiquement les valeurs des 4 constantes de raideurs, ainsi que de leurs distances d'actions.

4 Résultats

Il s'est avéré au cours des premières simulations, que les possiblités était beaucoup plus grandes qu'avec notre première approche par minimisation. Nous décrirons donc dans la suite plusieurs simulations que nous avons faites pour simuler différentes situations intéressantes.

Avant cela, nous allons présenter un paragraphe comparant nos deux approches, et ainsi montrer que dans les cas simples, permis par l'approchent par minimisation, nos deux démarches sont équivalentes.

4.1 Comparaison de nos deux approches

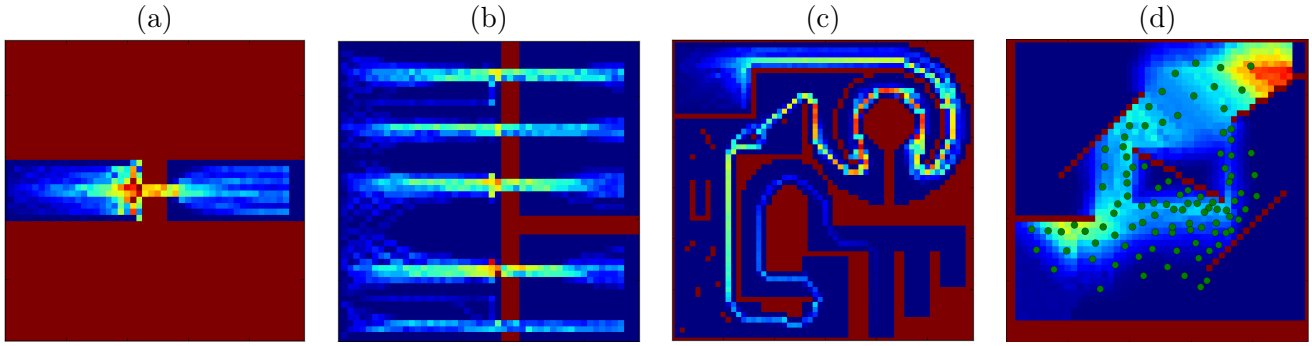

FIGURE 10: Comparaison des deux approches.(a)Carte de zones à risques à comparer avec la figure 5c.(b) Carte de zones à risques à comparer avec la figure 6c.(c)Carte de zones à risques à comparer avec la figure 8b. (d) Superposition de la carte de zone à risques obtenue avec la minimisation, et d'une itération du déplacement des billes

Après avoir repris les géométries utilisées dans les paragraphes 2.1,2.2 et 2.4, nous avons effectué des simulations avec les billes (dans les mêmes conditions), et tracé des cartes de zones à risques de la même façon que dans la section 2. Ces cartes sont présentées sur les figures 10a, b et c. La première chose à dire est ques les chemins empruntés par les billes sont équivalents. Cependant, on voit plusieurs différences. La première d'entre elles est la "non-homogénéité" des zones visités. Cela résulte de l'approche discrète plutôt que continue. On voit également comme prévu que les pions s'écartent après avoir passé les obstacles.

Ces cartes de zones à risques n'apportent rien de plus que celles calculées avec la minimisation, c'est pourquoi dans la suite nous présenterons d'autre types de résultats. Par exemple on a superposé sur la figure 10d une carte de zones à risques obtenue avec l'approche par minimisation et l'image d'une itération pendant le déplacement des billes. Ici on voit que les billes se déplacent sur les mêmes zones que la foule avec l'approche par minimisation. Bien sûr ici les billes représentent un seul instant t de la simulation, c'est pourquoi il y a une forte densité au milieu et non aux extrêmes (ce qui choque visuellement aux premier abord).

Ainsi, nous avons vu que les approches discrètes et continues sont équivalentes, du moins dans les situations calculables par minimisation. Les petites divergences montrent que l'approche discrète est plus performante. Dans la suite nous présenterons donc des simulations de différents cas, que nous n'aurions pas pu faire avec l'approche par minimisation, mais qui sont possibles avec cette seconde démarche.

4.2 Croisement de foules

Dans la partie minimisation, nous n'avons pas pu comparer nos résultats avec ceux de la communauté scientifique. Dans le cas des billes, nous avons eu la possibilité de le faire dans différents cas, par exemple celui du croisement de foules. La figure 11a présente les résultats issues de la thèse de P. Pecol [2]. Sur ceux là, on voit que la foule s'auto-organise en files indiennes. Nous avons réalisés la même experience, en mélangeant 500 personnes de deux populations différentes (250 bleux, et 250 rouges) lesquelles ont les mêmes caracteristiques, mais un objectif

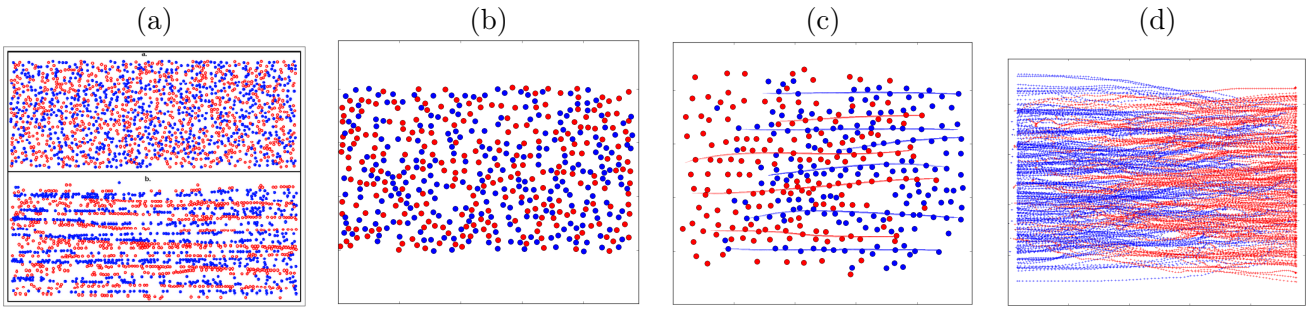

FIGURE 11: Croisement de foules. (a) Figure issue de la thèse de P. Pecol[2], avec 1500 piétons. (b) Conditions initiales : 250 bleus vont à droite et 250 rouges vont à gauche, les bleus vont à droite et les rouges à gauche. (c) Apparition des flux de personnes en lignes droites, on a superposé des lignes représentant les files indiennes. (d) Position de toute les billes à tous les instants : on voit que les "pacquets" bleux et rouges sont plus fins proche de la sortie, cela indique bien la concentration des individus dans des zones bien définies qui sont des lignes droites. Les files indiennes sautent moins aux yeux dans notre cas pour deux raisons. Premièrement, nous avons une plus courte distance à parcourir donc moins de temps pour se mettre en file indienne. Deuxièmement, notre interaction à grande distance empêche les individus d'être satisfaits en étant collés aux autres : dès qu'il y a de l'espace disponible, ils se réécartent un peu, phénomène que nous avons trouvé plus réaliste.

différent pour créer un croisement. La condition initiale est présentée sur la figure 11b, et les résultats sur les figures 11c et 11d. Les résultats sont probants puisqu'on observe comme prévu la création de files indiennes.

4.3 Evacuation d'un amphitéatre

De nouveau inspirés par la thèse de P. Pecol [2] dont les résultats sont présentés sur la figure 12a et 12b, nous avons nous aussi simulé l'évacuation d'une salle de classe (dans notre cas un amphitéâtre).

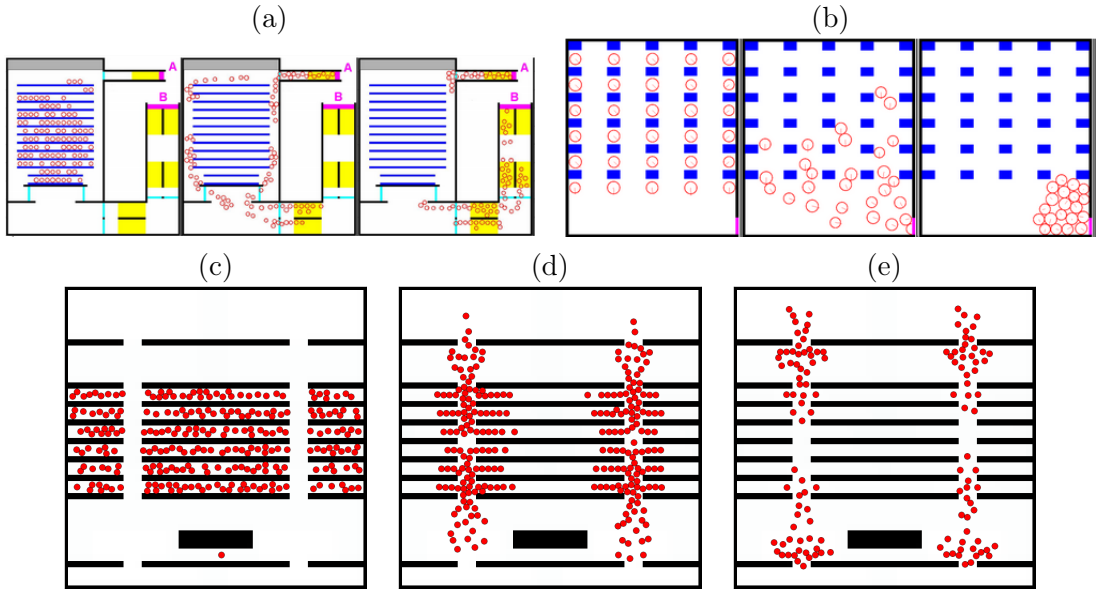

FIGURE 12: Simulation de l'évacuation d'une salle d'un amphitéatre contenant 251 personnes, avec prise en compte de "l'espace vitale".(a)(b) Résultats issues de la thèse de P. Pecol [2] pour une salle de classe standard et un amphitéatre. (c)(d)(e) Nos résultats, conditions initiales et évacuation en cours.

Dans ce cas, nos résultas divergent de ceux de P. Pecol. En effet, les individus évacuant la classe (fig 12b) s'entassent aux sorties, alors que dans notre cas (fig 12e), les gens restent écartés. Cela provient du fait que nous

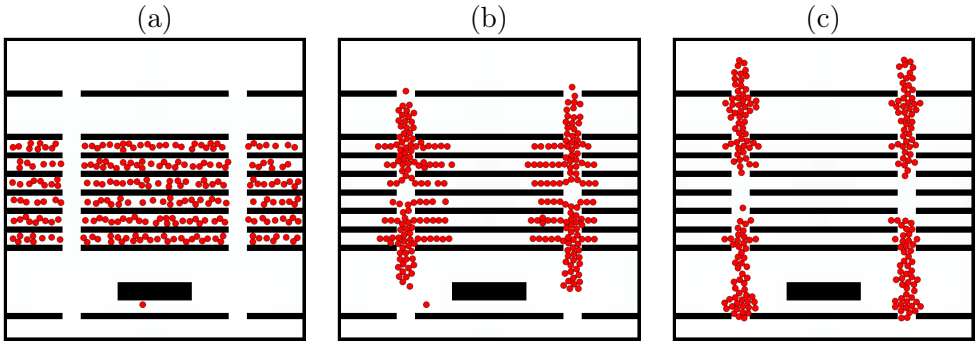

FIGURE 13: Simulation de l'évacuation d'une salle d'un amphitéatre contenant 251 personnes, sans prise en compte de "l'espace vitale".(a)(b)(c) Nos résultats, conditions initiales et évacuation en cours : on voit bien que les pions se serrent, et passent par la sortie par trois plutôt qu'un par un.

avons pris le parti de créer un modèle dans lequel les pions souhaitent conserver leur espace vital.

Si on considère la fin d'une heure de cours, notre modèle est donc consistant, puisque les gens prennent leurs temps pour sortir, sans se coller les uns aux autres. En effet, on voit bien les gens sortir des rangs en files indiennes, s'écarter pour ne pas être trop sérrés, puis se laisser passer les uns les autres et enfin évacuer l'amphitéatre un par un.

En revanche, notre modèle ne convient pas pour simuler une évacuation d'urgence, puisque les pions ne se poussent pas vers la sortie, comme le font ceux de P. Pecol.

Afin de confirmer cela, nous présentons sur la figure 13 le résultat d'une simulation dans les mêmes conditions, mais sans prendre en compte la notion d'espace vitale des individus. On voit clairement ici des résultats plus proches de la simulation de P. Pecol. En effet ici les pions se collent les uns aux autres, et évacuent beaucoup plus rapidement (82 itérations contre 115 avec l'espace vital). Ce cas est le seul pour lequel nous avons supprimé l'espace vital.

4.4 Une simulation à plus grande échelle : la gare de la Part-Dieu

Afin d'aller plus loin, nous avons testé notre programme sur une simulation à plus grande échelle. Pour ce faire, nous nous sommes inspiré d'un lieu réel : la gare de Lyon Part-Dieu. Pour cela, nous avons recréé l'architecture du batiment à partir du plan présenté dans l'annexe C. La Part-Ddieu compte trois points importants :

- La porte Rhône, par laquelle beaucoup de personnes entrent dans la gare. Cette porte est la plus empruntée, puisqu'elle est proche du métro. Les gens entrant par cette porte on pour but soit de prendre un train, soit de traverser la gare direction la porte Alpes.
- Cette porte Alpes, est très empruntée également, mais moins que la porte Rhône. Les gens passant par cette porte ont les mêmes but que ceux de la porte Rhône : traverser ou prendre un train.
- Enfin les voies d'accès aux quais par lesquelles les gens souhaitant prendre un train doivent passer.

Afin de modéliser au mieux le mouvement d'une foule dans un lieu comme celui ci, nous avons modifié quelque peu le programme de simulation. Dans la version précedente, dès lors qu'un pion arrivait au niveau d'une sortie, celui ci disparaissait de la simulation, et était considéré comme "évacué". Ici, nous avons vu les choses autrement : dès qu'une personne franchit une sortie, on lui attribut automatiquement une nouvelle sortie. Ainsi la bille semble rebondir, et cela simule un "chassé-croisé" des individus par une porte. Le pion qui sort prend donc l'identité d'une personne qui rentrerait exactement à ce moment là.

Sur la figure 14a, on présente la géométrie, sur laquelle sont indiquées en rouge les sorties. Afin de réattribuer les sorties, nous avons dû définir (dans ce cas) 6 matrices distances. La simulation se déroule donc comme ceci : La bille se dirige vers la sortie i avec une probabilité P_i ; dès que celle-ci a traversé la sortie, on lui attribue une nouvelle sortie (et donc une nouvelle matrice D) j avec une probabilité Pj.

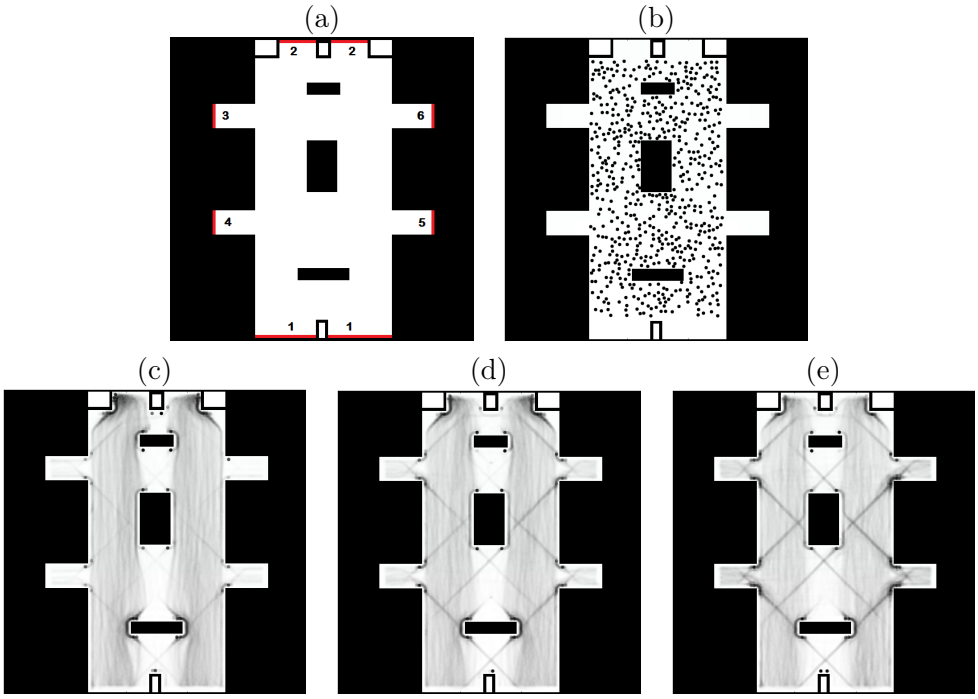

FIGURE 14: Simulation de l'évolution d'une foule de 600 personnes dans une architecture comme celle de la gare de la Part-Dieu. (a) Géométrie inspirés de la gare de Lyon Part-Dieu. (b) Conditions initiales : 600 personnes. (c)(d)(e) Superposition de toutes les itérations avec $P_{quais} = 0.1, 0.25 et 0.5$.

La condition initiale est présenté sur la figure 14b : 600 personnes réparties aléatoirement dans la gare de la Part-Dieu.

Il y a beaucoup de phénomènes très intéressant à étudier dans ce genre de configurations. La création de files dans les couloirs à gauche et à droite en est un très bon exemple. Ces créations de files sont difficiles à traduire en figures, et les vidéos présentés ci-jointes se passent de commentaires. Nous avons donc décider de présenter ici un résultat intéressant que nous pouvons simplement et graphiquement présenter : l'influence des probabilités des différentes sorties sur les chemins empruntés.

Comme dit plus haut, la plupart des gens se trouvant à la Part-Dieu ne font que traverser. La porte la plus empruntée est la porte Rhône : la sortie 1. La porte Alpes est également très empruntée, mais un petit peu moins que la précédente, celle ci correspond à la sortie 2. Enfin les 4 accès aux quais son peu empruntés : ce sont les sorties 3 à 6.

Nous avons effectué 3 simultations, dans lesquelles nous avons fait varié les probabilités d'accès aux quais, afin de voir l'impact de ces probabilités. Nous avons résumé dans le tableau 2 ci dessous les probabilités de chaques sorties.

Sortie / **Simulation**	**1**	**2**	**3**	**4**	**5**	**6**
1 (figure 14c)	0.56	0.40	0.01	0.01	0.01	0.01
2 (figure 14d)	0.53	0.37	0.025	0.025	0.025	0.025
3 (figure 14e)	0.50	0.30	0.05	0.05	0.05	0.05

TABLE 2: Probabilités utilisés pour les différentes acquisitons

En se servant du logiciel de traitement d'images ImageJ, nous avons effectué une superposition de toutes les images (fonction Z Project) pour chaque acquisiton. Les résultats sont présentés sur les figures 14c, 14d et 14e. De prime abord, on voit bien que les zones les plus visitées sont les couloirs à gauche et à droite. Cela signifie que la

foule évite bien les obstacles centraux. Ensuite, on peut voir l'apparition de lignes obliques, plus foncées. Ces lignes représentent les chemins empruntés par les individus de la foule pour aller prendre leurs trains, ou pour sortir de la gare après un long voyage en train. On voit clairement que ces lignes deviennent plus foncées si l'on augmente la probabilités des quais. Cela indique que les gens ont plus emprunté ces voies. Ce qui est intéréssant de voir est que ces lignes ne sont pas perturbées par le flux d'individus circulant entre les deux portes principales. De même les flux de personnes ne sont pas altérés par ces chemins secondaires, en effet, le fond diffus dans les couloirs gauche et droit sont homogènes. Cela nous montre que les flux de passagers voulant rejoindre leur trains préferent rester sur le chemin optimal, et bien que génés par le flux de passager continu dans des directions différentes, ils suivent bien ce chemin optimal.

Cela nous confirme une nouvelle fois que cette notion de matrice distance est la clef de voûte de cette approche, et que celle-ci conditionne entièrement le déplacement des individus dans une géométrie donnée, même en la présence d'un nombre très important de personnes.

Nous avons effectué d'autre types de simulations, mais les résultats étant très visuels, et difficilement comparables à des résultats existants, nous n'avons pas trouvé pertinent de les faire figurer dans le rapport.

Conclusion et perspectives

Durant ce projet, nous avons implémenté deux méthodes pour simuler des mouvements de foules. La première, classique et abondamment présente dans le domaine, dont le principe est de modéliser les individus par des pions se repoussant par des forces répulsives. L'autre, plus innovante, qui modélise la foule par une densité continue se déplaçant dans une matrice discrète par un calcul de minimisation.

Il est important de noter plusieurs choses. Premièrement, les deux méthodes donnent des résultats pertinents. Cependant, notre approche par minimisation est beaucoup plus limitée que l'approche plus classique. En effet, cette approche permet uniquement de faire se déplacer une foule compacte dans un espace donné, et elle ne renvoie pas de résultats quantifiables précisément. Elle permet tout de même d'avoir accès à une carte de "zones à risques" assez facilement, dont la teneur est proche de ce que l'on attendait. Concernant l'approche discrète (avec les pions), on peut dire que l'on a trouvé exactement ce que l'on attendait. La foule se déplace comme attendu (résultat qualitatif), nous avons retrouvé des résultats connus comme la création de files indiennes lorsque deux foules se croisent. Nous avons cependant ajouté un nouveau paramètre par rapport aux simulations que l'on peut trouver sur internet. Nous avons ajouté une petite répulsion à plus grande distance qui simule l'espace vital des individus. Cet ajout a fait ses preuves notamment dans le réalisme de la foule, l'exemple le plus marquant étant certainement la réorganisation de la foule après les files indiennes de croisement. Les individus ne restent pas en file indienne, ils se replacent comme on pourrait s'y attendre dans la réalité.

Si nous devions résumer notre travail en quelques lignes, nous dirions que pendant ce projet nous avons fait deux choses complètement différentes. La première, tester notre intuition sur un modèle inexistant. Ce fut un travail de recherche compliqué mais très intéressant. Ce modèle n'est pas meilleur qu'un autre mais nous avons réussi à avoir des résultats pertinents par nos propres moyens (l'algorithme A* mis à part). La deuxième, implémenter un modèle connu, avec un travail de recherche personnelle moindre. Celui-ci nous a permis d'avoir un programme fonctionnel sur une plus grande palette de situations, et surtout de vérifier les résultats de notre premier modèle. En effet, nous avons comparé la carte de "zones à risques" de l'approche par minimisation avec le programme "billes", et avons pu constater que les deux approches se valaient parfaitement (dans les limites des capacités de la minimisation).

Enfin, voilà les astuces que nous pourrions retenir dans le cadre d'un projet numérique :

— Etudier le projet au préalable et être clair sur les objectifs à atteindre.

— Être propre dans l'écriture des programmes, et dans la gestion des dossiers/versions dès le départ.

— Tester les morceaux de code un par un avant de tout assembler.

— Faire comme si une personne ne connaissant pas le programme allait le lire juste après, afin de ne jamais se perdre dans notre propre programme, même si on ne le regarde plus pendant un moment.

— Être sûr de là où l'on veut aller et ne pas abandonner devant un problème apparemment très compliqué, mais savoir changer d'approche si le modèle utilisé est à bout de souffle (dans notre cas, après moult tentatives pour réanimer notre programme de minimisation et après l'avoir emmené en terre inconnue, nous n'avons pas voulu tomber dans le piège de l'acharnement thérapeutique et avons décidé de le laisser partir en paix...).

— Une fois les objectifs atteints, tester, comparer et trouver les limites du modèle.

— Discuter des perspectives que pourraient offrir le modèle.

Dans notre cas, les perspectives (autrement dit ce que nous aurions fait si le projet avait duré plus longtemps) sont simples. Le programme de minimisation était très intéressant à implémenter, mais objectivement il ne fait pas avancer le problème. Nous pourrions reprendre le programme "billes" et ajouter une prise de décision pour les individus (par exemple anticiper et éviter à l'avance les zones noires de monde). C'est cet élément qui pourrait faire passer le modèle de "représentatif" à "impressionnant de réalisme". Nous pourrions alors simuler non pas seulement des évacuations, mais tout type de mouvements de foules, et caractériser beaucoup plus précisément le mouvement dans l'espace et dans le temps (flux locaux, régulation de flux de personnes, etc.).

Annexes

A Gestion des obstacles par la minimisation

Cette partie devait se trouver initialement dans la partie 2, sur les réusltats obtenus avec l'approche par minimisation. Cependant, les résultats etant difficiles à interprétés, nous avons préferé les placer en annexe.

Nous avons montré des situation comportant des murs "longs". Nous pouvons également créer des géométries comportant des obstacles ponctuels disposés le long de chemins, ou de couloirs.

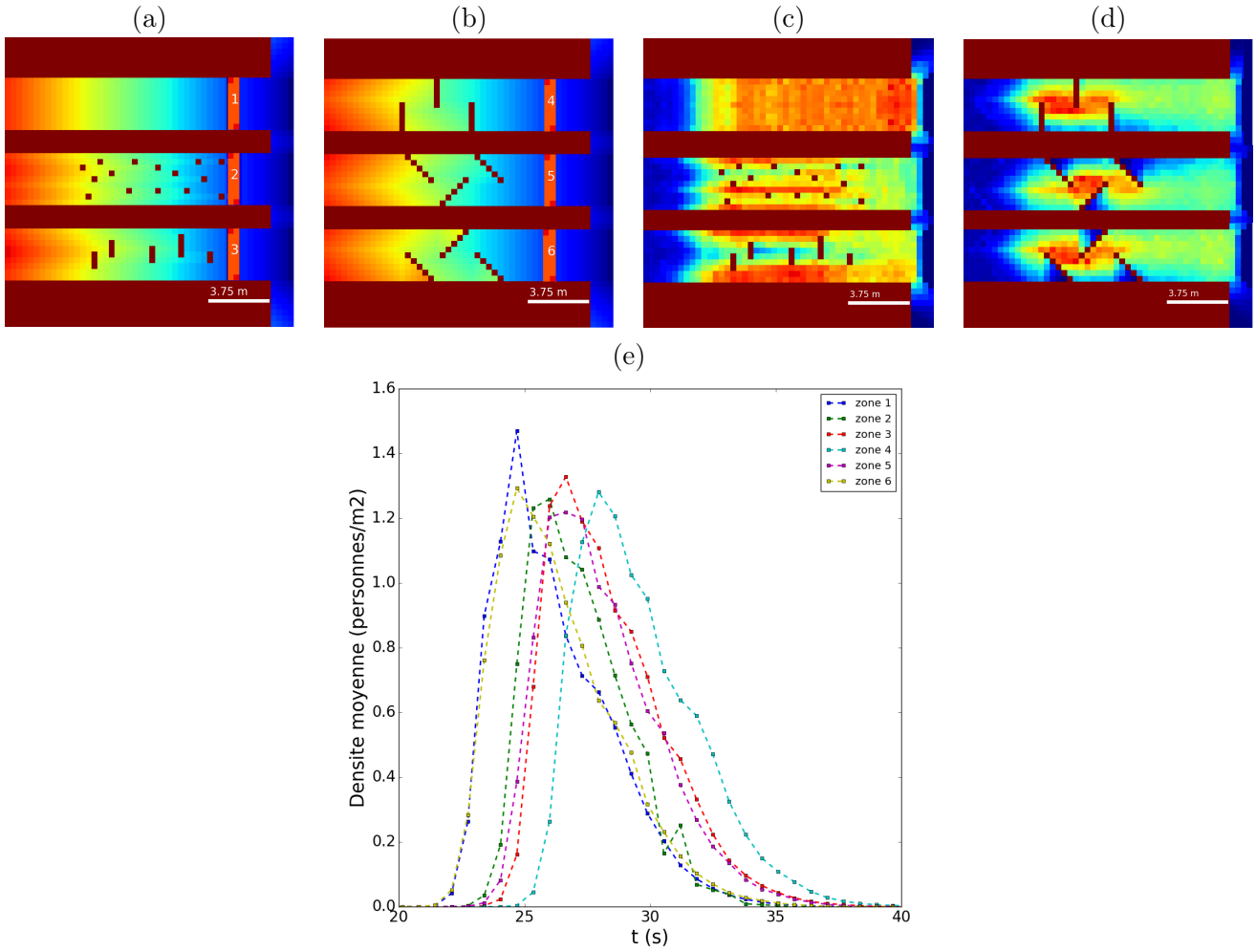

FIGURE 15: Comparaison d'obstacles.(a) Première matrice, et emplacement des zones 1 à 3.(b) Deuxième matrice, et emplacement des zones 1 à 3. (c) Carte des zones à risques pour les couloirs 1 à 3. (d) Carte des zones à risques pour les couloirs 4 à 6.(e) Densités dans les zones 1 à 6.

Pour cela nous avons créé deux matrices (figures 15a et 15b) comportant en tout 6 couloirs. Ceux-ci sont identiques en tout point, et on comme référence le couloir 1 qui est vide. Nous avons ensuite placé dans chacun des autres couloirs des obstacles de différentes natures, afin de constater de l'effet de ces obstacles sur la vitesse de la foule. Chaque couloir (excepté le premier) comporte au total 15 cases d'obstacles. Au temps t=0, nous avons placé 50 personnes dans 15 m^2 à l'extrémité gauche de chaque couloir, puis éxecuté le programme de simulation de foule dans chacun des cas. Ensuite, nous avons tracé les profils de densité dans chacune des zones après le passage de la foule (figure 15e). A noté que les conditions initiales sont rigoureusement les mêmes pour les 6 acquisitions, ce qui permet de comparer les temps de passage de la foule.

Le premier résultat qui est satisfaisant, est que la foule évite correctement les obstacles, et que ceux-ci ont bien un effet sur le déplacement de la foule. En effet on peut voir sur les cartes de zones à risques (figure 15c et 15d) que

les zones les plus visitées sont différentes en fonction de la géométrie des couloirs. De plus les profils de densités sont décalés sur l'axe des temps les uns par rapport aux autres, certaines géométrie ont plus ralenti la foule que d'autres et c'est cela que nous allons voir en détail maintenant. En effet, en comparant le début et la fin du profil de densité, on peut connaitre le temps de passage du début et de la fin de la foule dans la zone voulue.

Premièrement, nous pouvons voir que chacune des géométries mettant en jeu des obstacles a ralenti la foule par rapport au couloir vide (à l'exception du cas de la zone 6 dont nous reparlerons à la fin de ce paragraphe).

Ensuite, il est clair que la situation la plus défavorable pour la foule est la situation 4 (profil de densité très décalé par rapport au profil du couloir 1). Dans ce cas, la foule fait face à 3 murs normaux à la direction de déplacement priviligié. Ce résultat est cohérent avec une situation réèlle dans laquelle des gens doivent éviter de tels murs, et donc être très ralentis. On voit également sur la carte de zone à risque qu'entre les deux premiers murs, la densité est très supérieur à celle dans le reste du couloir, cela montre l'agglutinement des personnes dans un petit périmètre.

On peut voir ensuite que les couloirs 3 et 5 sont à peu près équivalent du point de vue du ralentissement de la foule. Cela est étonnant puisque dans le 3, les individus ont préféré se séparer et longer les murs alors que dans le 5 ils sont concentrés au centre. Il semblerai donc qu'avec notre technique de déplacement de foule par minimisation, ces deux géométries sont équivalentes.

Les obstacles dans le couloir 2 apportent un fait intéressant. En effet, on voit que la foule s'est regroupé en files indiennes plus concentrées afin d'éviter les obstacles. On voit également que certains individus ont été obligés de quitter ces files et ont ainsi contourné les obstacles, cela est dû à la répulsion des individus entre eux. Ces résultats sont en accord avec l'observation quotidienne, où on peut effectivement voir des files de personnes se créer pour éviter des obstacles avec cependant certains individus exclus de la foule, et obligés de slalomer entre les obstacles.

Enfin, le cas de la zone 6 est intéressant également. De prime abord nous avons pensé que c'était la géométrie du système qui fournissait un résultat prometteur, car donnant exactement le même temps qu'un couloir vide. Cependant, il est très difficile de savoir si ces résultats (qualitativement plus poussés) sont représentatifs de la réalité ou s'ils ne sont que le fruit d'imprécisions non maîtrisées. Il est alors clair que nous ne pouvons pas pousser l'analyse plus loin que cela.

En conclusion sur cette comparaison de cas, il est clair que les obstacles ont un réel impact sur le déplacement de la foule. En revanche, dans un cas un petit peu plus subtil, l'approche par minimisation a montré ses limites, limites que nous retrouverons dans le cas suivant.

B Complément à la partie 4.1

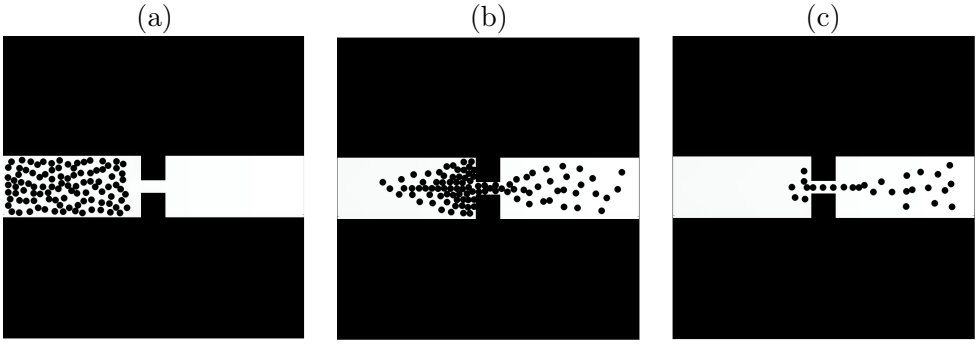

FIGURE 16: Images issues de l'animation du déplacement des billes (figure 10a).(a) Condition initiale.(b) Les individus attendant pour pouvoir rentrer dans le petit couloir se collent contre les murs pour ne pas être trop sérrés. (c) Après l'obstacle les pions s'écartent, on voit aussi que les gens en fin de foule sont plus écartés des autres, ceci afin de maximiser leur espace vital.

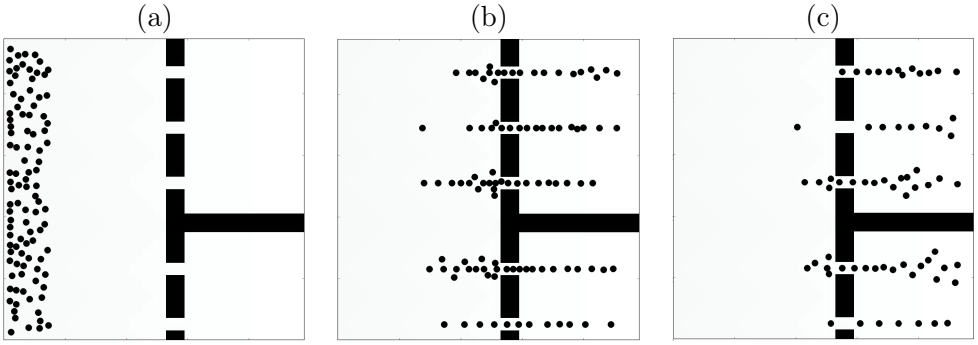

FIGURE 17: Images issues de l'animation du déplacement des billes (figure 10b).(a) Condition initiale.(b) Les individus arrivent aux portes. (c) Après l'obstacle les pions s'écartent, on voit aussi les gens se coller contre le métro en attendant leur tour.

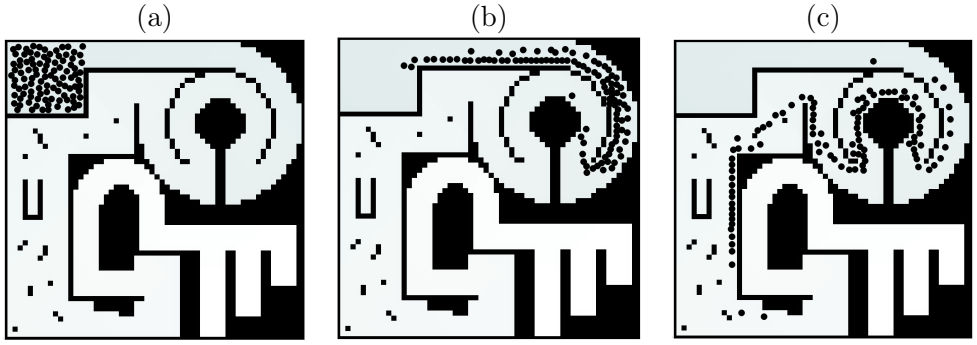

FIGURE 18: Images issues de l'animation du déplacement des billes (figure 10c).(a) Condition initiale.(b)Un déplacement difficile dans le "rond-point". (c) Après plusieurs obstacles, le tri a été fait : les pions les plus rapide sont passés devant, et on observe ainsi une file indienne.

C Complément à l'étude de la gare de la Part-Dieu

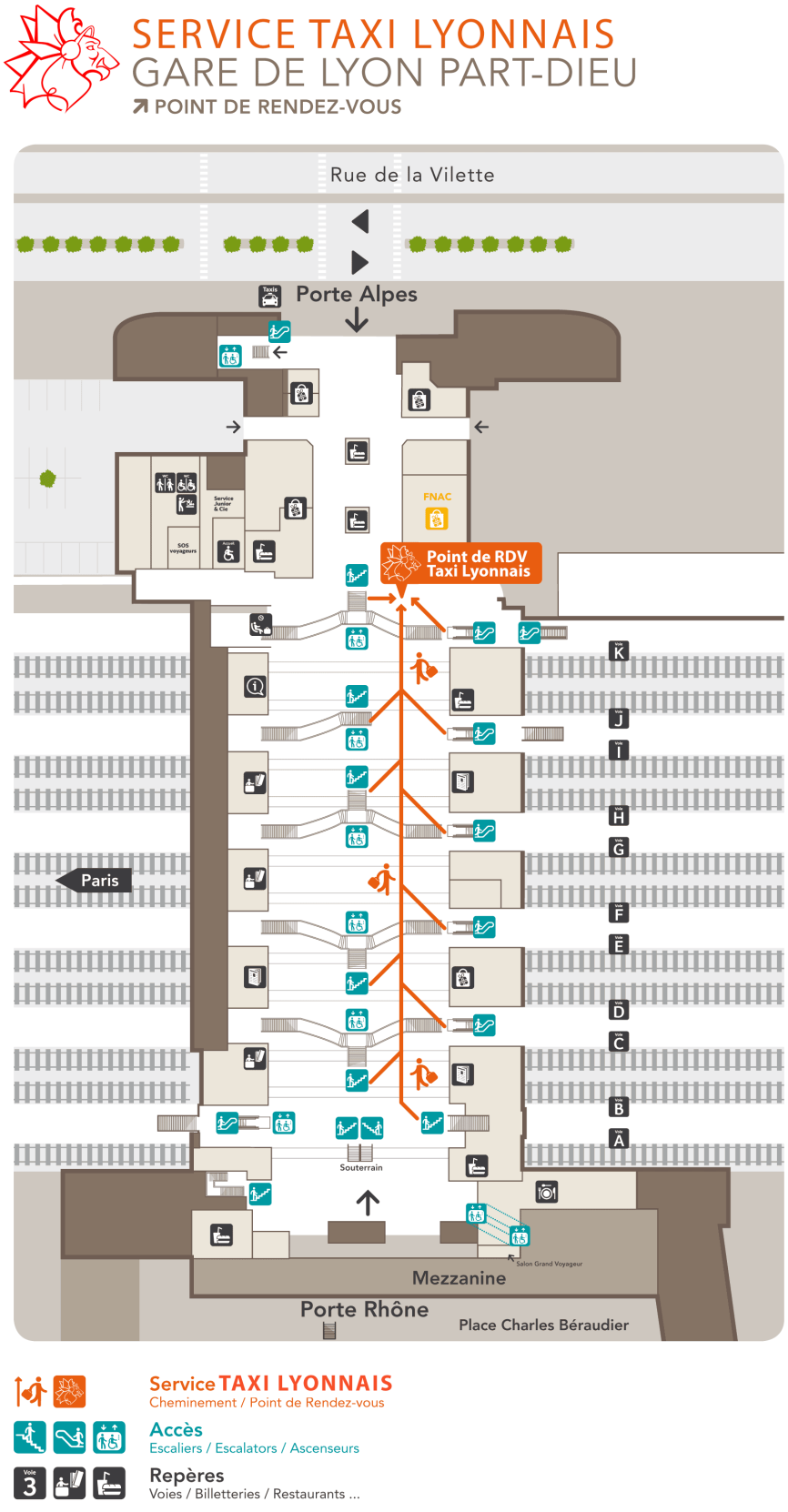

FIGURE 19: Plan de la gare de la Part-Dieu.

D Notice d'utilisation des programmes

D.1 Création de la matrice distance

Avec une matrice dessinée dans paint

- Dans paint (ou un de ses cousins) on crée un nouveau fichier et on redimensionne l'image pour la taille de matrice voulue (50×50 est un bon choix, attention les programmes de simulation n'acceptent que les matrices carrées). On dessine du noir aux endroits où l'on veut mettre des obstacles, et du rouge pour ce qui concerne les sorties. Une fois la matrice prête, on l'enregistre sous le nom **geometrie.png** dans le même dossier que le programme de simulation utilisé.
- Rien de plus simple, et c'est déjà fini.
- Maintenant on lance le programme Matrix_D.py.
- On choisit le mode 'à partir d'une image' dans le choix au début du programme (choix 0).
- Ensuite, on choisit si l'on veut utliser cette matrice pour l'approche par minimisation, ou avec les billes. (La différence est un paramètre dans l'algorithme de recherche de chemin.) On peut choisir l'option "minimisation" si on veut une matrice représentative pour les deux approches.
- Plusieurs figures s'affichent, pas de panique il n'y rien à faire à part les fermer.
- On laisse A-star faire son boulot. Une fois fini (une fois l'affichage arrivé à 100%), la matrice résultante s'affiche et s'enregistre toute seule. On peut fermer le programme et passer à la suite.

Avec le mode intéractif Avec cette méthode de création de matrice, il faut faire attention au nombre de cliques que l'on fait pour créer les obstacles. En effet un certain nombre de cliques maximal est permis par le module que nous avons utilisé. Il est donc important de noter que le mode intéractif devra êter utilisé pour des matrices relativement simples. Ex : un labyrinthe droit passe car le nombre de murs est limité. Mais si on veut commencer à faire des formes arrondies, il faut dessiner avec paint (c'est d'ailleurs pour cela que nous avons trouvé un autre moyen de dessiner nos matrices).

- On lance le programme Matrix_D.py.
- On choisit le mode intéractif au début du programme (choix 1).
- Ensuite, on choisit si l'on veut utliser cette matrice pour l'approche par minimisation, ou avec les billes.
- On rentre la taille de la matrice (50 est toujours un bon choix) que l'on veut dessiner.
- On rentre 1 ou 0 selon si on veut mettre des obstacles ou pas.
- Si on rentre 1, une figure s'affiche avec une matrice vierge. Le fonctionnement est simple. On ne peut faire que des murs droits, et pour chaque mur, il faut cliquer aux deux extrémités. On clique donc à la suite pour tous les murs que l'on veut, et on ferme la figure quand on a fini. Si par malheur un nombre impair de cliques était fait, le dernier ne serait pas pris en compte.
- Rebelote, mais cette fois pour des obstacles ponctuels. On dit oui ou non, si oui une figure s'ouvre et on clique sur toutes les cases où l'on veut voir un 'plot'. Subtilité : le clique gauche pose un plot, le clique droit enlève un plot. Enfin, quand on a terminé, on ferme la figure.
- Une autre figure se rouvre immédiatement, il ne faut pas la fermer. Il faut cliquer sur les sorties (de la même manière que les murs, les sorties sont des barres droites entre deux points où l'on a cliqué). Quand on a fini, on ferme la figure.
- Le programme affiche la version finale qui est envoyée à A-Star. On ferme la figure et on laisse le programme tourner.
- Le programme affiche la matrice distance calculée, on ferme la figure et on est prêts pour la suite.

Une fois que le fichier 'Ultimate_Matrix.txt' a été créé avec succès, on le déplace dans le dossier dans lequel se trouvent les outils pour effectuer la simulation (continue ou discrète).

D.2 Programme Foule_Matrix.py, approche continue.

Quand tout fonctionne.

- Une fois que l'on a notre matrice distance sous la forme 'Ultimate_Matrix.txt' dans le dossier où se trouve le programme Foule_Matrix.py, on peut lancer ce dernier.
- La première chose demandée est la population, c'est à dire le nombre de gens que l'on veut dans la matrice (typiquement 100 personnes).
- Ensuite une figure s'affiche et il faut cliquer pour placer la où l'on veut des foules. On clique en haut à gauche et en bas à droite de chaque rectangle où l'on veut mettre une foule.
- On peut faire ainsi plusieurs rectangles à la suite, mais il est très déconseillé d'en faire plus d'un car le programme est optimisé pour une seule foule.
- Il est aussi très conseillé de ne pas faire de rectangles trop grands. 15 cases sur 15 cases est bien un maximum si on veut que le temps d'exécution soit relativement court.
- Une figure récapitulative s'ouvre, on peut rester à l'admirer mais il vaut mieux la fermer pour que le programme démarre.
- Enfin, on laisse le programme tourner. A chaque itération il affiche le nombre de gens qu'il reste dans la matrice, ce qui donne une idée d'où en est le programme
- Lorsque le programme a fini de tourner il affiche l'animation du mouvement. Si l'on veut qu'il enregistre à chaque itération l'image de l'itération, il faut décommenter une ligne dans la boucle animation.

Quand on a des échecs. On peut avoir des échecs (le programme l'affiche dans le terminal). Dans ce cas, le programme recommence du début avec des conditions initiales légèrement différentes. Bien sûr, il est probable que le programme recommence à faire des échecs. Dans ce cas :

- On regarde si en reessayant le programme y arrive.
- Si oui, et bien c'est bon.
- Si non, on peut reesayer en adoucissant la demande de minimisation. Par exemple en baissant le nombre de personnes dans la matrice. Si ça ne marche toujours pas, la géométrie est probablement trop dure pour l'approche continue..

D.3 Programme billes.py, approche discrète

- Une fois la matrice 'Ulimate_Matrix.txt' disponible, on lance le programme.
- Le programme nous demande successivement si on veut insérer une population de type 1, 2, et 3.
- Les caractéristiques de ces populations (rayon, vitesse, couleur) sont modifiables dans le programme.
- Si on dit oui (rentrer 1 sinon 0), une fenêtre s'affiche et comme d'habitude on clique pour déterminer les endroits des foules. En haut à gauche puis en bas à droite du rectangle où l'on veut placer une foule. On fait ces deux cliques pour chaque foule que l'on veut successivement (ici, il est vivement conseillé de mettre plusieurs foules : more foules = more fun), puis on ferme la figure.
- Le programme demande alors combien de billes il doit mettre dans chaque rectangle que l'on a défini. Il faut avoir en tête les nombres que l'on va donner au programme à chaque rectangle que l'on clique.
- ATTENTION ! Le programme place les gens dans les rectangles aléatoirement, en évitant de les placer dans des murs ou de les superposer. S'il n'y a pas assez de place pour accueillir tout le monde, le programme va décider combien de personnes il peut mettre dans la zone.
- Le programme fait ça pour les trois populations, et ensuite il se lance dans le calcul.
- A chaque itération, il affiche le nombre de billes restantes dans la matrice. Encore une fois cela sert à voir la rapidité de calcul et le nombre de gens qu'il reste, donc le temps qu'il reste avant la fin de l'éxecution.
- Le programme s'arrête lorsque le nombre d'individu est inférieure au nombre fixé (variable 'fin_boucle' dans le programme). Si le programme tourne indéfiniment avec un nombre de personne fixe, il faut modifier 'fin_boucle', car cela veut dire que des personnes sont bloquées dans la géometrie.

— Lorsque le programme a fini de tourner il affiche l'animation du mouvement. Si l'on veut qu'il enregistre à chaque itération l'image de l'itération, il faut décommenter une ligne dans la boucle animation.

D.4 Resultats et traitement

Pour chacune des deux approches, lorsque le calcul est fini, il affiche le résultat sous forme d'une animation. Durant cette animation, le programme enregistre dans un sous dossier /resultats/save_tableaux les résultats à chaque itération. Si l'on veut aller plus loin, on peut lancer le programme traitement.py

— Celui-ci demande tout d'abord le nombre de fichiers. Il faut lui indiquer le nombre de fichiers présents dans le sous dossier /resultats/save_tableaux. (Ce nombre de fichiers correspond également au nombre d'itérations de la simulation)

— Une fois le chargement terminé, la carte de zone à risques s'affiche, après l'avoir admiré, on peut la fermer (celle-ci s'enregistre également toute seule dans le sous dossier /traitement).

— Ensuite, une nouvelle carte de zones à risque s'affiche. Sur celle-ci on peut cliquer sur des zones (comme on le fait avec le placement des foules). Ces zones servent à calculer et afficher la densité dans la zone en fonction du temps.

— Le plot de la densité en fonction du temps s'affiche, on peut le fermer.

— Pour l'approche continue : La densité maximale et moyenne au cours du temps s'affichent ensuite, puis le traitement est fini.

— Pour l'approche discrète : Une nouvelle animation du mouvement se lance, et pendant cette animation le programme sauvegarde les frame dans le dossier /resultats. (Il est important de noter que pour le traitement, tous les individus sont équivalents, on ne différencie plus les foules.)

E Architecture de l'archive

L'archive contient :

— Le rapport

— Un readme

— Un dossier Acquisitions, dans lequel on trouve toutes les acquisitions étudiées dans le rapport. Chaque acquisiton s'accompagne du programme permettant de faire la simulation (tous les paramètres sont rentrés par défaut). Ce dossier contient également un dossier vidéos, dans lequel on a stocké des vidéos d'acquisitions effectuées avec le programme billes.py. Dans cette section, les programmes ne sont pas *propres*, les programmes commentés se trouve dans le dossier Programmes.

— Un dossier Programmes, dans lequel on retrouve d'un côté l'ensemble les outils pour créer la matrice distance, puis les outils pour simuler le déplacement d'une foule avec l'approche par minimisation et enfin un dernier dossier dans lequel se trouvent les outils pour simuler la foule à l'aide de l'approche discrète.

F Liste des vidéos

— Croisement de foules : Acquisitions/videos/croisement.avi

— Sortie d'amphitéatre : Acquisitions/videos/amphi_cool.avi

— Evacuation d'amphitéatre : Acquisitions/videos/amphi_evac.avi

— Gare de la part dieu : Probabilité des quais 1% : Acquisitions/videos/gare_quai_proba1.avi, Probabilité des quais 2.5% : Acquisitions/videos/gare_quai_proba25.avi, Probabilité des quais 5% : Acquisitions/videos/gare_quai_proba5.avi

— Evacuation de la salle des terminaux du CBP : Acquisitions/videos/cbp.avi

Références

[1] FB36 (MIT). A* algo, Dec 2010. http ://code.activestate.com/recipes/577519-a-star-shortest-path-algorithm/.

[2] Phillipe Pecol. *Modelisation 2D discrete du mouvement des pietons.* PhD thesis, Universite Paris-Est, 2012.

[3] universcience.tv. L'intelligence des mouvements collectifs. https ://www.youtube.com/watch ?v=ptn0pHwSdAE/.