Compte rendu mini projet
Simulation de foules

Encadré par Thomas Chatagnon

A travers ce compte rendu, nous allons vous présentez comment nous avons été confrontés au développement de la simulation de foule posée par Thomas Chatagnon, doctorant à l'INRIA-Rennes. En effet, nous avons comme objectif de mettre en œuvre un simulateur de foules humaines d'après le modèle de « Force sociale » développé par Helbing et Molnar en 1998 [1]. Une fois cette tâche réalisée, nous avions la possibilité d'améliorer le code existant.

I) Présentation du travail existant

Le but de ce projet est de se familiariser avec l'utilisation de code open source et de maîtriser la lecture de fichier input (fichier d'entrée), l'utilisation d'un solveur et la création de fichier output (fichier de sortie). Le modèle, fourni par M. Chatagnon, auquel nous étions confrontés est un classique de l'étude de la dynamique de foule pouvant générer différents scénarios et permettant de visualiser les résultats grâce à un outil de visualisation. Tout le code source est en Python et chaque individu seront appelé par le terme d'agent.

1) Lecture des fichiers « Input »

Pour pouvoir faire une simulation de foules, il faut d'abord paramétrer la position initiale de chaque agent, leur vitesse moyenne, leur taille, ainsi que la forme de leur environnement et leurs objectifs. Toute ces informations sont fournies dans des fichiers *.txt* : permet de paramétrer la position de chaque mur en donnant leurs deux extrémités ; *parameter_templete.txt* permet de paramétrer le rayon moyen, la vitesse maximale moyenne, et la déviation standard que chaque agent a avec ces valeurs ; *goals_template.txt* permet de paramétrer la position des objectifs que les agents vont devoir atteindre, ces objectifs peuvent prendre la forme de points ou de segments ; enfin *agents_positions_template.txt* permet de déterminer la position des agents au début de la simulation sous deux formes différentes : soit l'on donne la position de chaque agents individuellement directement, soit on donne la position des coins d'une zone dans lesquelles un nombre n d'agents apparaissent aléatoirement.

Afin de lire ces fichiers, il existe dans le script *in_out_class.py* une classe *read* dans laquelle est définie les différentes fonctions permettant de lire ces fichiers et de renvoyer les informations importantes sous la forme que l'on veut.

2) Solveur de la simulation du mouvement de la foule

La simulation se fait avec la fonction *run_script()* dans le script *Solver.py*. Cette fonction commence par appeler les fonctions définit plus tôt pour importer les informations nécessaires à la simulation et crée ensuite les vecteurs de la position de chaque agent, leur vitesse, leur vitesse maximum, leur rayon, la position des murs, la position des objectifs, et enfin une matrice *Checkpoints* contenant l'information de si un certain agent a déjà atteint un certain objectif.

Une boucle s'exécutant jusqu'à ce que tous les agents ont atteints leurs objectifs est ensuite créer. Dans cette boucle, les forces agissant sur chaque agent actif est calculer puis utiliser pour calculer sa nouvelle vitesse et donc sa nouvelle position. Les forces qui s'appliquent sur chaque agent sont : une force de répulsion des murs, une force de répulsion des agents, et une force d'attraction du prochain objectif.

3) Création de fichiers « Output »

Comme dit précédemment, à chaque pas de temps, on récupère différentes données utiles comme les positions des agents, leurs vitesses… Pour récupérer ses données, on ne va pas se contenter d'afficher les résultats sur la console de Python, on souhaite avoir ses données sur des fichiers *.txt*. Pour cela, la classe *output* dans le script *in_out_class.py* défini des fonctions permettant de créer des fichiers *.txt* et d'y écrire directement les paramètres que l'on souhaite.

Ainsi, on crée au début de l'exécution du solveur, un fichier *.txt* et on y écrit les positions et les vitesses initiales de chaque agent. Puis, pour chaque agent et à chaque pas de temps de la simulation, on y écrit sa position, sa vitesse et l'instant t de la simulation.

4) Récupération des données et animation

Maintenant que le solveur est mis en place, il est temps d'animer cette simulation de foule grâce au script *Animation.py* majoritairement basé sur la librairie *matplotlib*. Ce script va d'abord tracer les murs en lisant les le fichier *walls_positions_template.txt*, placer fictivement les différents objectifs, placer les agents en les représentants par des cercles de rayons prédéfinis dans le fichier *parameter_templete.txt* puis enfin définir les limites spatiales de la simulation.

Enfin, le script va animer la simulation en affichant, pour chaque pas de temps de simulation, les positions des agents, récupérer grâce la lecture du fichier *.txt* créer via la classe *output*. Une représentation de l'animation est présentée en figure (1).

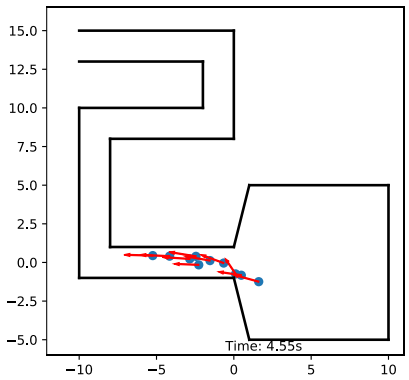

Figure 1 : exemple d'animation de simulation de foule

II) Modification des fichiers « Input »

La première modification que l'on a fait est la façon dont on donne la position initiale des agents : initialement nous ne pouvions pas créer plusieurs groupes d'agents différent ayant des objectifs différents, tous les agents avaient les mêmes objectifs et nous ne pouvions créer qu'une seule zone d'apparition pour les agents. Nous avons donc cherché à modifier la structure des fichiers d'entrée de manière à pouvoir générer un nombre quelconque de groupe d'agents ayant des positions initiales et des objectifs différents.

1) A quoi ressemble les nouveaux fichiers « Input »

Nous avons décidé de combiner les fichiers *agents_positions_template.txt* et *goals_template.txt* en un seul fichier *group_template.txt*. Ce fichier contient d'abord une liste des objectifs existants, prenant la forme soit de un point, soit de deux, en fonction de si l'objectif en question est un point ou un segment. Le fichier contient ensuite la mention *random* ou *manual* en fonction de si la prochaine partie correspond à des agents placer aléatoirement ou manuellement. Si on place les agents aléatoirement, chaque ligne suivant la mention *random* contient, dans cet ordre, la position du premier coin de la zone d'apparition, puis du deuxième, puis du nombre de personne apparaissant dans cette zone d'apparition et enfin la liste des objectifs que ces agents vont suivre désignée par leur indice. Si la mention *manual* est présente, chaque ligne contiendra la position d'un agent puis la liste des objectifs qu'il suivra.

2) Modification de la classe *read*

Nous avons ensuite dû combiner les deux fonctions responsables de la lecture des fichiers précédents en une fonction *read_group()*. Cette fonction, en plus de devoir renvoyer le vecteur des position initiales et la liste des objectifs, doit renvoyer la matrice *Checkpoints* qui est créer de sorte que les objectifs qu'un agent ne doit pas atteindre soit déjà considérer comme étant atteint.

La lecture du fichier se fait grâce à une boucle *while* qui parcourt le fichier jusqu'à la mention de *random* ou *manual* après quoi il rentre dans une autre boucle dans laquelle il va lire les informations correspondantes. Ces informations sont ensuite utilisées pour créer le vecteur *Position* et la matrice *Checkpoints*.

Des modifications minimales ont été nécessaire sur la fonction *run_script()* pour qu'il utilise la nouvelle fonction qui a été définie.

III) Modification du modèle d'expression de force

Une autre modification de ce script passe par une modification du solveur et non plus par une modification des fichiers *txt* et des différentes classes. On se propose maintenant une modification du solveur et des forces dans une optique de comparaison de modèle.

1) Prise en compte de la dépendance angulaire

Dans le script, seul l'expression de la force répulsive des murs est issue du modèle de 1998, l'expression de la force répulsive des agents est une version amélioré développé par Helbing et Johansson en 2013[2]. De plus, le modèle de 2013 prend en compte la dépendance angulaire des agents. Ainsi, il est défini l'angle de dépendance entre l'agent i et l'agent j :

$$\varphi_{ij} = \frac{\boldsymbol{v_i}}{\|\boldsymbol{v_i}\|} \cdot \frac{-\boldsymbol{d_{ij}}}{\|\boldsymbol{d_{ij}}\|} \quad (1)$$

Avec $\boldsymbol{v_i}$ le vecteur vitesse de l'agent i et $\boldsymbol{d_{ij}}$ le vecteur distance pointant de l'agent j à vers l'agent i.

Ainsi, il est défini le préfacteur $\omega(\varphi_{ij})$des interactions de forces :

$$\omega(\varphi_{ij}) = \left(\lambda_i + (1-\lambda_i)\frac{1+\cos\varphi_{ij}}{2}\right) \quad (2)$$

Avec λ_i un paramètre augmentant avec la force des interactions. Une optimisation du paramètre faite dans le travail de Helbing et Johansson[2] donne $\lambda_i \approx 0.1$.

Dans le solveur, on apportera donc une fonction *prefactor* prenant en arguments la vitesse de l'agent i ainsi que la position de l'agent i et de l'agent j et qui renvoie le préfacteur ω et on ajoute ce préfacteur à notre force d'interaction des agents (voir Annexe).

2) Modèle de Karamouzas

Le modèle utilisé est, comme énoncé en préambule de ce compte rendu, le modèle de « Force Sociale » développé par Helbing et Molnar en 1998 [1]. Cependant, on souhaite comparer ce modèle, que l'on généralise par le modèle d'Helbing, avec un autre modèle, le modèle de Karamouzas de 2014[3]. Ce modèle se base sur une estimation probabiliste que deux agents se croisent dans un certain temps τ et les faits se repousser si cette probabilité est haute. Une force simulée entre deux agents peut donc être déduit de cette estimation.

$$\boldsymbol{F}_{ij} = -\left[\frac{k e^{\frac{-\tau}{\tau_0}}}{\|\boldsymbol{v}_{ij}\|^2 \tau^2}\left(\frac{2}{\tau}+\frac{1}{\tau_0}\right)\right]\left[\boldsymbol{v}_{ij} - \frac{\|\boldsymbol{v}_{ij}\|^2 \boldsymbol{x}_{ij} - (\boldsymbol{x}_{ij}.\boldsymbol{v}_{ij})\boldsymbol{v}_{ij}}{\sqrt{(\boldsymbol{x}_{ij}.\boldsymbol{v}_{ij})^2 - \|\boldsymbol{v}_{ij}\|^2\left(\|\boldsymbol{x}_{ij}\|^2 - (R_i+R_j)^2\right)}}\right] \quad (3)$$

En pratique, ce modèle va encourager les agents à se déplacer parallèlement les uns aux autres, mais va avoir du mal à marcher dans des situations où les agents sont forcés de se croiser.

3) Autre ajouts.

En plus des forces existantes, nous avons rajouter des forces de contact définie par Helbing en 2000[4] servant à empêcher que deux agents, ou un agent et un mur, se trouve littéralement l'un sur l'autre. Cette force dépend linéairement de la distance entre les deux agents, est égale à 0 quand les deux agents se frôlent et maximale quand les deux agents se trouve sur le même point, avec la valeur maximale dépendant d'un facteur k choisi de sorte à avoir des résultats réalistes.

Nous avons aussi changé le fichier *paramater_template.txt* de sorte à inclure les valeurs de k et τ_0 utilisée dans le modèle de Karamouzas ainsi qu'un paramètre pour définir si on utilise le modèle de Helbing ou celui de Karamouzas.

Nous avons modifié le solveur de sorte qu'il enregistre dans un fichier texte la densité dans un certaine zone et la vitesse moyennes des agents à chaque instant.

IV) Résultats

L'un des scénarios que l'on a essayé de simuler est un scénario ou deux groupes de personnes se croisent dans un couloir de sorte à reproduire les conditions d'une étude expérimental faite en 2017 par Shuchao Cao[5]. En faisant cette simulation, nous nous sommes rendu compte d'un problème : les agents restaient bloquer contre les murs alors qu'ils ont juste besoin de passer le coin du mur pour aller vers leur objectifs (voir figure 2). On s'est rendu compte que c'était à cause de la combinaison des deux murs qui a un certain point avait une force de répulsion égale à la force d'attraction de l'objectif. Pour régler le problème nous avons dû réduire la taille du segment objectif au bout du couloir afin qu'il la force d'attraction de l'objectif de soit pas aussi perpendiculaire au mur.

A partir de ces simulations nous avons tracer la densité dans l'intersection en fonction de la vitesse moyenne des agents afin de reproduire les graphs montrer dans l'étude de Shuchao Cao (voir figure 3).

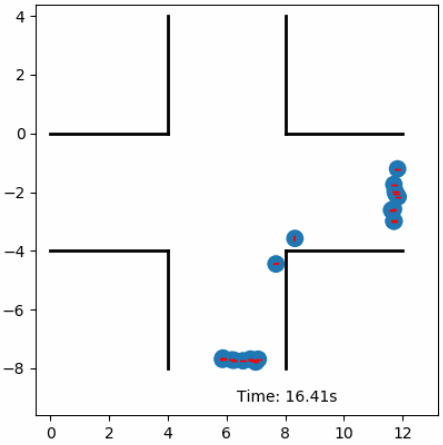

Figure 2 : Les agents se coincent contre les murs

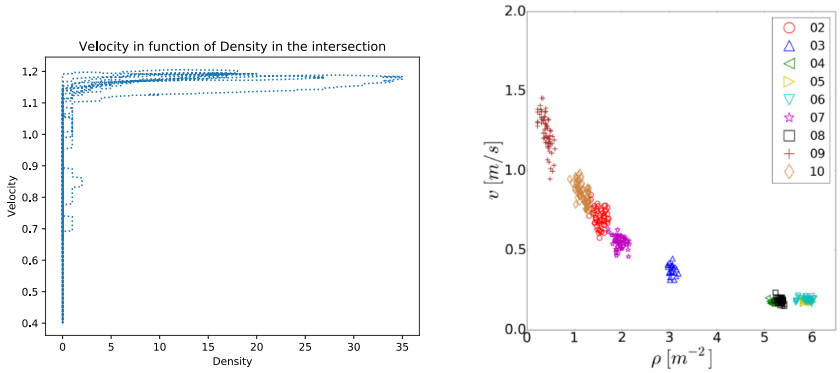

Figure 3 : Vitesse en fonction de la densité, nos résultats à gauche et les résultats de Shuchao Cao à droite

V) Conclusion

Nous avons ainsi créé un script capable de faire des simulations de foule en nous basant sur des modèles existants. Nous avons pu observer certains problèmes dans ces simulations qui sont dû aux modèles simplistes sur lesquels nous nous sommes basés. Ces simulations restent néanmoins un bon moyen de simuler des comportements réels dans des cas simples. Par manque de temps nous n'avons pu faire que très peu d'expérimentation avec ces simulations mais de nombreux scénarios ont le potentiel d'être étudier dans le futur.

Référence Bibliographiques :

[1] : Social force model for pedestrian dynamics – Hebing and Molnar – 1998

[2] : Pedestrian, Crowd, and Evacuation Dynamics – Helbing and Johansson – 2013

[3] : Universal power law governing pedestrian interactions – Karamouzas, Skinner and Guy – 2014

[4] : Simulating Dynamical Features of Escape Panic – Helbing, Fakas and Vicsek – 2000

[5] : Fundamental diagrams for multidirectional pedestrian flow – Cao, Seyfried, Zhang, Holl and Song – 2017

Resource

https://github.com/Francokab/Simulation-de-foule