Etude de la dynamique des foules

Par le biais de la physique moderne

Auteurs :

NADAUD Antonin
JANINI Raphaël
LUBIN Thomas
NUCE LAMOTHE Augustin

Encadrant :

DESPLAT Lucie

11 mai 2025

CY Tech – préing-2 groupe 2

Sommaire

1 Objectif

1.1 Situation 1

La première situation (imposée) dans le cadre de cette étude, consiste à modéliser l'évacuation d'une foule depuis un espace rectangulaire, tel qu'une salle. Les individus se dirigent vers une sortie sous l'effet d'une force motrice, tout en interagissant les uns avec les autres par le biais de forces sociales. Chaque individu est associé à une vitesse cible, représentant son intention de déplacement. L'objectif est d'avoir le temps d'évacuation totale pour la foule. Dans cette simulation le nombre de personne est fixé à 45.

1.2 Situation 2

On repart sur les bases de la situation une, mais cette fois dans deux salles de cour. Les deux salles sont exactement identique à la différence que la salle 1 a une porte et que la salle 2 a deux portes. Le nombre d'individu (étudiant + éléve) est de 31. Notre objectif est de voir si la porte en plus est vraiment utile.

2 Théorie sous-jacente

Le modèle de force sociale, introduit par Helbing et Molnár [1], vise à simuler le comportement de piétons dans des environnements denses, en représentant chaque individu comme une particule soumise à des forces comportementales. Deux forces fondamentales structurent ce modèle : la force motrice et la force sociale.

Force motrice. Chaque individu i cherche à atteindre une vitesse souhaitée $\vec{v}_i^0$ dans une direction donnée. Ce comportement est modélisé par une force motrice, qui pousse l'individu à ajuster sa vitesse actuelle $\vec{v}_i$ à sa vitesse désirée, selon :

$$\vec{f}_i^{\mathrm{m}} = m_i \frac{\vec{v}_i^0 - \vec{v}_i}{\tau_i} \tag{1}$$

où m_i est la masse de l'individu et τ_i est un temps de relaxation caractérisant la rapidité avec laquelle l'individu tente d'atteindre sa vitesse cible.

1. https://doi.org/10.1103/PhysRevE.51.4282

Force sociale. Outre la volonté individuelle de se diriger vers une destination, les piétons interagissent entre eux via des forces dites sociales, traduisant leur tendance à maintenir une distance interpersonnelle. Ces interactions sont modélisées par une force répulsive de la forme :

$$\vec{f}_{ij}^{\,\mathrm{s}} = A_i \exp\left(\frac{r_{ij} - d_{ij}}{B_i}\right) \vec{n}_{ij} \tag{2}$$

où :
- B_i est une constantes positives liée à la zone de confort de la personne i,
- r_{ij} est la somme des rayons corporels des individus i et j,
- d_{ij} est la distance entre les centres de masse des deux individus,
- $\vec{n}_{ij}$ est le vecteur unitaire pointant de j vers i.

Ces forces permettent de simuler des comportements réalistes d'évitement et de gestion de l'espace dans des environnements contraints, comme lors d'une évacuation.

3 Méthode

3.1 Determiner l'équation

Afin d'établir l'équation, nous avons tout d'abord supposé que le référentiel d'étude est galiléen, pour ensuite appliquer la seconde loi de newton $\sum \vec{F}_{\text{particule}} = m_{\text{particule}}\vec{a}$.

On a :

$$\vec{f}_{\text{direction}} + \vec{f}_{\text{social}} = m_{\text{particule}} \frac{d\vec{v}}{dt}.$$

Au final après quelques simplification on a :

$$\frac{\vec{v}_i^{0} - \vec{v}_i}{\tau_i} + \frac{1}{m_{\text{particule}}} \sum_{j \neq i} \exp\left(\frac{r_{ij} - d_{ij}}{B_i}\right) \vec{n}_{ij} = \frac{d\vec{v}}{dt} \tag{3}$$

3.2 Resolution d'équation différentiel

3.2.1 Contextualisation

Pour résoudre (3). Nous avons décider d'utiliser méthode d'euler :

$$\vec{v}_{n+1} = \vec{v}_n + \Delta t \cdot \vec{F}_n$$

On introduit une personne modélisée en Python par un dictionnaire (pour mieux comprendre la fonction qui permet de résoudre cette équation) :

```
{
        "position": np.array([0, 0]),
        "masse": 10,
        "vitesse_desiree": 1.34,
        "vitesse": np.array([0, 0]), #vitesse initiale
        "to": .2,
        "rayon": 10 + random.randint(-2, 2),
        "destination": np.array([100,100])
}
```

- `"position"` : Un tableau NumPy qui contient les coordonnées $[x, y]$ de la personne dans l'espace.
- `"masse"` : La masse de la personne, ici fixée à 10 (arbitrairement).
- `"vitesse_desiree"` : La vitesse désirée que la personne souhaite atteindre, (ici 1.34 ms^{-1}).
- `"vitesse"` : Un tableau NumPy qui représente la vitesse initiale de la personne. La vitesse initiale est définie comme un vecteur nul $[0, 0]$ (immobile au départ)
- `"to"` : Un paramètre fixé à 0.2. Il pourrait représenter un coefficient de friction, de résistance ou tout autre facteur de modification des interactions.
- `"rayon"` : Le rayon de la personne (representée comme un cercle), calculé comme 10 plus un nombre aléatoire compris entre -2 et 2.

3.2.2 Présentation du code

Voici un exemple pour resoudre l'equation différentiel suivante (force motrice) :

$$\frac{\vec{v}_i^0 - \vec{v}_i}{\tau_i} = \frac{d\vec{v}}{dt} \tag{4}$$

Le code qui permet de resoudre donc l'equation differentiel :

```
1  def euler(tab_personne, personne,indice, step=.02):
2
3
4      f_totale = force_motrice(personne) #cacul de la force motrice
5
6      #projection sur Ux et Uy
7      vitesse_x =  personne["vitesse"][0] + step * f_totale[0]
8      vitesse_y = personne["vitesse"][1] + step * f_totale[1]
9
10     #on actualise la position
11     personne["position"] = np.array( [
12         personne["position"][0] + vitesse_x,
13         personne["position"][1] + vitesse_y
14     ])
15
16     # v(t_n+1)
17     personne["vitesse"] = np.array([
18         vitesse_x,
19         vitesse_y
20     ])
```

Dans les premières lignes on calcul la force totale.
Puis on applique la méthode d'euler :

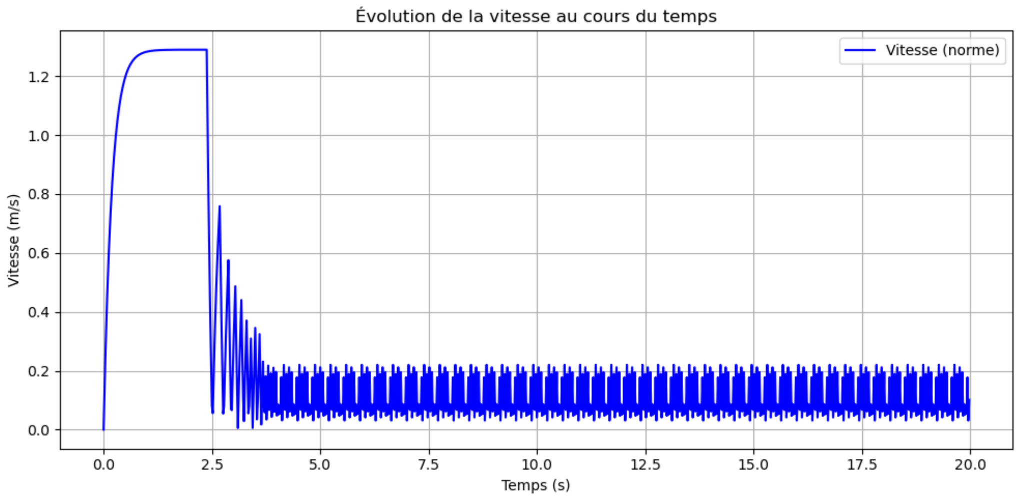
$$\vec{v}_{n+1} = \vec{v}_n + \Delta t \cdot \vec{F_{totale}} \tag{5}$$

Pour chaque direction (0_y) et (0_x) on projette pour avoir la vitesse en composante respective x et y. (ligne 8 et 9), ici $\Delta t = 0.02$ ce qui minimise la marge d'erreur.

La dernière ligne permet d'actualiser $\vec{v_n}$.

3.2.3 Résultat graphique

On lance le progamme, sur une particule on recupère les diverses valeurs de V_n pour en suite tracer la courbe si dessous (la paricule démarre à la position $(0, 0)$ et vise le point $(100, 100)$:



Le résultat est bien cohérent on voit que la particule atteint à la manière d'une exponentielle inversé sa $v_{desiree}$ pour en suite l'atteindre totale. Dès que la position $(100, 100)$ est atteinte sa vitesse décroit pour revenir vers la position. On voit qu'en suite la fonction devient périodique, la particule passe d'un bout à l'autre de la position (100,100) sans jamais l'atteindre.

Méthode de Runge-Kutta d'ordre 2

Cette méthode plus couteuse temporellement s'avére être plus précise qu'euler.

L'équation générale est :

$$\vec{v}_{n+1} = \vec{v}_n + k_2$$

avec :

$$k_1 = h \cdot \vec{F}(\vec{v}_n, \vec{r}_n)$$

$$k_2 = h \cdot \vec{F}\left(\vec{v}_n + \frac{1}{2}k_1, \vec{r}_n\right)$$

Dans le code :

— Le pas de temps est $h = 0{,}02$

— `k1` est calculé par :

$$k_1 = h \cdot \texttt{resultante(tab\_personne, personne, indice, obstacles, portes)}$$

— Puis on modifie temporairement la vitesse :

$$\vec{v} \mathrel{+}= \frac{1}{2}k_1$$

— Ensuite, on recalcule la force et :

$$k_2 = h \cdot \texttt{resultante(...)}$$

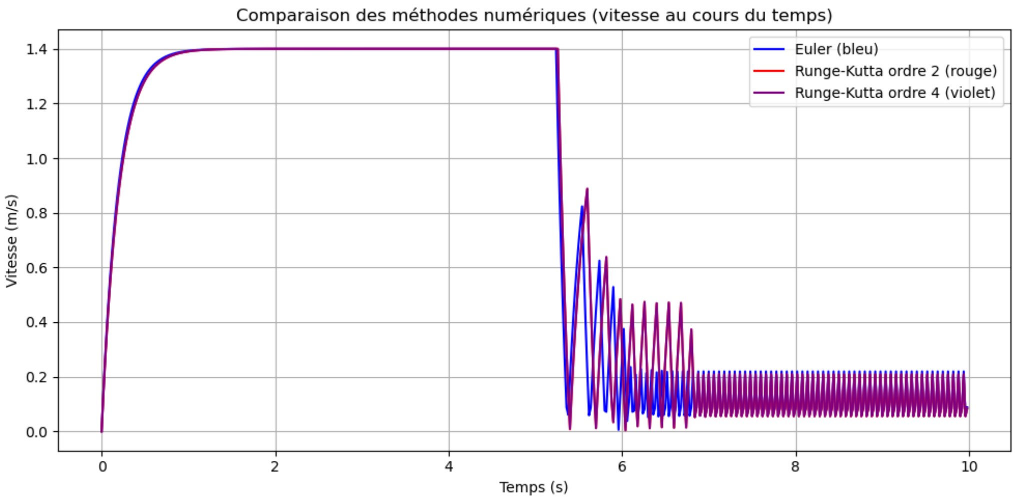
— Enfin, on met à jour la vitesse avec :

$$\vec{v}_{n+1} = \vec{v}_n + k_2$$

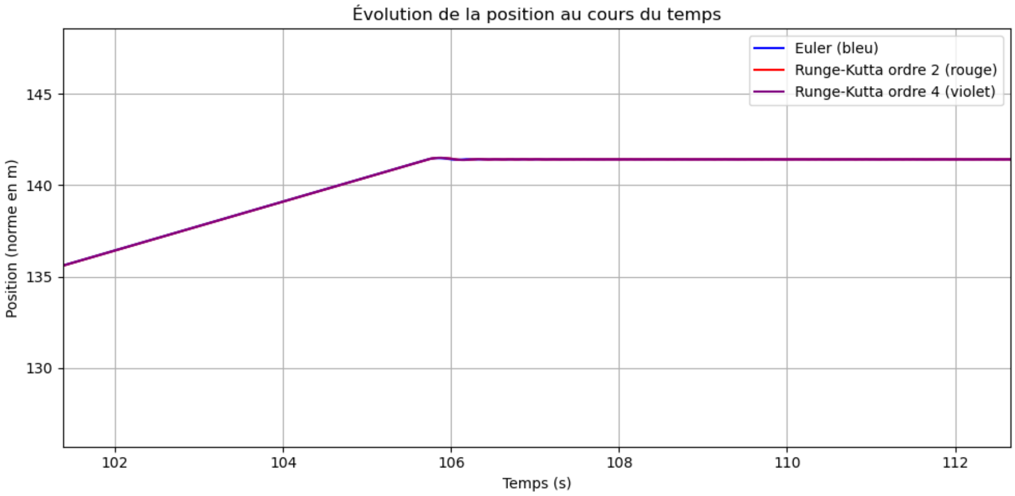
(voir code)

```
def runge_kutta_2(tab_personne, personne,indice,obstacles,
    portes, step=.02)
```

3.2.4 Resultats de rungge kutta 2 et 4



Au final, on observe un léger écart, dû à la plus grande précision de la méthode de Runge-Kutta. Mais cette écart sur la vitesse n'a pas un grans impact sur la position à en croire le graphique suivant :



4 Explication du code

4.1 Structure et contextualisation

Le code est divisé en trois modules distincts :

— **main.py** s'occupe de dessiner la fênetre de gérer plusieurs actions il initialise le tableau de personne (tableau de particule)

— **physique.py** contient uniquement des fonction pour la gestion physique

— **affichage.py** contient les situations et centralise la gestion des dessin dans le canvas

4.2 Modélisation physique

Cette section présente les différentes fonctions du fichier **physique.py**, accompagnées de leur signature pour faciliter leur identification dans le code.

4.2.1 Force motrice

Pour caculer la force motrice on applique simplement (1)

```
def force_motrice(personne):
```

La fonction **calcul_i0** retourne le vecteur directionnel en soustrayant le point souhaité de la position actuelle de la personne, puis en normalisant le résultat comme suivant.

Soient a la position de la particule et $ptSouhaite$ les coordonnées de la porte :

$$\vec{e_\theta} = \vec{a} - pt\vec{Souhaite} \tag{6}$$

4.2.2 Force social(s)

```
def force_intercation_social(tab_personne, personne, indice,
↪  b0=config["b0"], seuil_interaction=50)
```

Pour caculer on applique (2)

Afin de limiter la complexité temporelle de la simulation, on ajoute deux conditions : une personne ne peut pas interagir avec elle-même (ce qui est logique d'un point de vue physique), et elle n'interagit qu'avec les autres personnes situées à moins de 50 unités de distance. Ce seuil d'interaction (fixé à 50) est vérifié physiquement : au-delà de cette distance, l'influence sociale devient négligeable. A noté que cette valeur à été fixé expérimentalement.

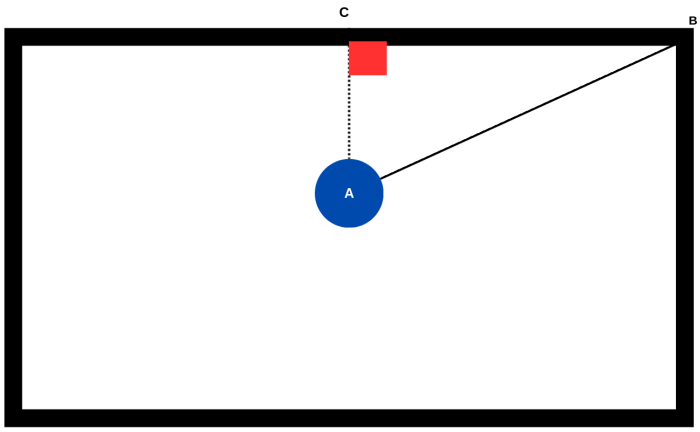
Pour calculer d_{ij} onprend la norme de la position de la personne avec la norme de la personne avec laquel elle intéragis on soustrait en suite les deux rayon qui correspond à r_{ij}. $\vec{n_{ij}}$ est le vecteur normal (a - b) qu'on norme pour avoir un vecteur unitaire.

4.2.3 Force répulsion mur

```
def force_interaction_social_mur(personne, indice, portes, b0
↪  = config["b0"]):
```

La force de repulsion de mur est plus compliquée à calculer. Il faut la calculer pour chaque mur avec toujours un seuil d'intéraction (un mur agis sur nous que s'il nous touche). r_{ij} reste le même. Mais pour caculer d_{ij} cette fois ci on utilise pythagore.

La condition sert à ne pas agir si on a une porte. La fonction `distance_mur_vect` utilise le théorème de Pythagore. Elle renvoie un tuple avec, premièrement, la distance entre le mur visé ($\mathbf{d_{ij}}$), ainsi que le vecteur normal ($\mathbf{n_{ij}}$). Pour mieux comprendre voici un exemple :



On a déjà la longueur AB et on cherche AC, on obtient facilement l'angle $\widehat{ABC}$ que l'on notera α.
On a alors : $BC = \sin(\alpha) \cdot AB$

Finalement d'après le théorème de pythagore $\sqrt{AB^2 - CB^2} = AC$, pour le vecteur normal on en déduit les coordonnés de c pour en suite normaliser $\vec{CA}$.

Cette opération est répétée pour les autres murs de manière casi analogue. Une condition supplémentaire est ajoutée afin d'ignorer la zone correspondant à l'emplacement de la porte.

4.2.4 Force interaction rectangle

```
def force_intercation_rectangle(personne, rectangle,
↪ b0=config["b0"]):
```

Pour cette méthode on a créé une fonction qui détermine le point le plus proche d'un rectangle. Selon la méthode suivante

Vecteurs de Projection : Soit un rectangle défini par ses coordonnées de coin inférieur gauche (x, y), sa longueur l, et sa hauteur h. Nous définissons deux vecteurs :
— $\vec{v}_{\text{longueur}} = (l, 0)$: vecteur représentant la longueur du rectangle.
— $\vec{v}_{\text{hauteur}} = (0, h)$: vecteur représentant la hauteur du rectangle.

Position Relative : La position de la personne par rapport au coin inférieur gauche du rectangle est :

$$\vec{p}_{\text{relatif}} = (x_{\text{personne}} - x, y_{\text{personne}} - y)$$

Projection sur les Bords du Rectangle : Pour trouver le point le plus proche sur le rectangle, on projette $\vec{p}_{\text{relatif}}$ sur les vecteurs $\vec{v}_{\text{longueur}}$ et $\vec{v}_{\text{hauteur}}$. On a donc :

$$k_1 = \frac{\vec{p}_{\text{relatif}} \cdot \vec{v}_{\text{longueur}}}{\|\vec{v}_{\text{longueur}}\|^2}$$
$$k_2 = \frac{\vec{p}_{\text{relatif}} \cdot \vec{v}_{\text{hauteur}}}{\|\vec{v}_{\text{hauteur}}\|^2}$$

Ajustement des Projections : Les valeurs de k_1 et k_2 sont ajustées pour s'assurer qu'elles se situent dans le rectangle :

— Si $k_1 < 0$, alors $k_1 = 0$.
— Si $k_1 > l$, alors $k_1 = l$.
— Si $k_2 < 0$, alors $k_2 = 0$.
— Si $k_2 > h$, alors $k_2 = h$.

Calcul du Point le Plus Proche : Le point le plus proche sur le rectangle est donc :

$$\vec{p}_{\text{proche}} = (x, y) + k_1 \cdot \frac{\vec{v}_{\text{longueur}}}{\|\vec{v}_{\text{longueur}}\|} + k_2 \cdot \frac{\vec{v}_{\text{hauteur}}}{\|\vec{v}_{\text{hauteur}}\|}$$

Pour finir on applique (2) ici $r_{ij} = r_i$ - 0 car pas de rayon.

4.3 Resolution

Situation 1

Bilan des forces appliquées : Trois types de forces principales agissent sur l'individu :

— **Les forces motrices**, modélisées par l'équation (1) ;
— **Les forces d'interaction entre personnes**, données par (2) ;
— **Les forces d'interaction avec les murs**, également représentées par (2).

On peut également intégrer les interactions avec d'autres obstacles. Dans le cas où aucun obstacle n'est présent, le vecteur de force associé sera simplement nul : $\vec{0}$.

Ainsi, on peut généraliser le système via une fonction basée sur la méthode d'Euler (et de Runge-Kutta), capable de traiter l'ensemble des cas évoqués.

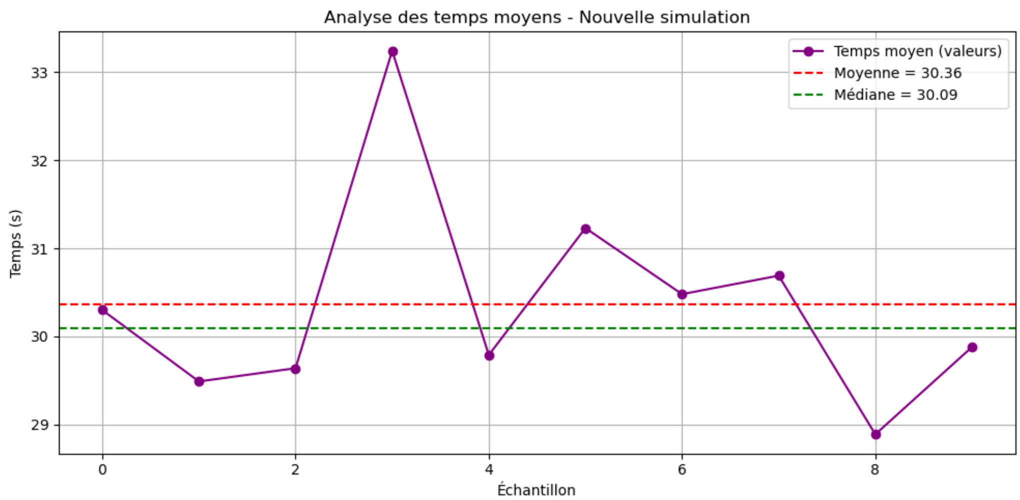
C'est ce que font les fonctions de résolutions suivantes.

```python
def runge_kutta_2(tab_personne, personne,indice,obstacles, portes, step=.02):
def runge_kutta_4(tab_personne, personne,indice,obstacles, portes, step=.02):
def euler(tab_personne, personne,indice,obstacles, portes, step=.02):
```

5 Analyse

Le temps de notre expérience est en unités de temps il ne dépend pas de la puissance de l'ordinateur pour un résultat plus fiable c'est seulement un incrément à chaque étape de la modélisation. Les différences observées sont dû à la vitesse qui varie de $1.34 \pm 0.25 m.s^{-1}$

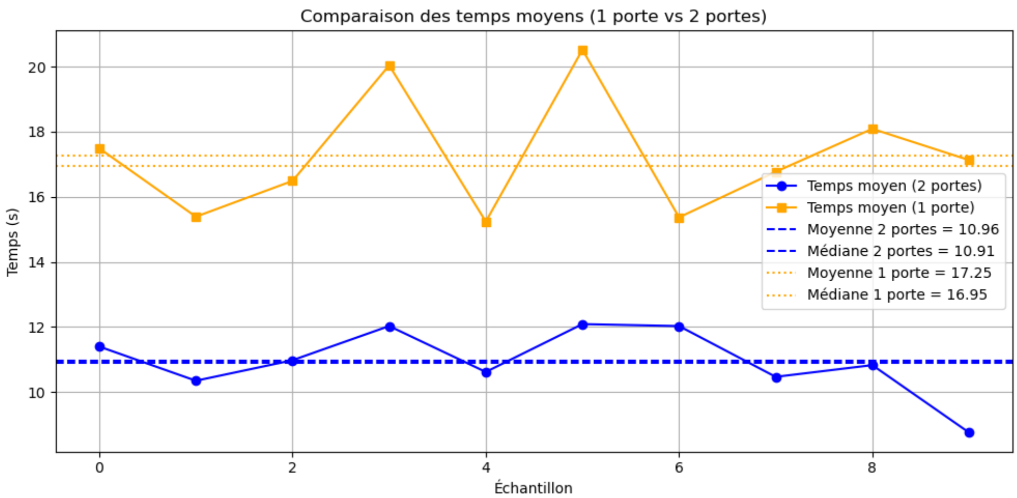
5.1 Situation 1



Sur un échantillonage de 10 valeurs

Pour cette simulation on obtient un temps moyen de **30.36 unités de temps**.

5.2 Situation 2



On obtien une moyenne de temps sur 10 échentillons pour la classe avec 2 portes de 10.96 unité de temps tandis qu'avec 1 porte la moyenne est de 17.25 unité de temps. Pour ramener sur une echelle plus parlante que des unités de temps dans le cas ou il y a seulement une porte le temps sera augmenté d'environ **61%**.

Conclusion Une telle augmentation peut avoir des conséquences critiques en situation d'urgence, car chaque seconde supplémentaire augmente le risque de blessure, en particulier en cas de panique ou de mouvements de foule désorganisés.

A noter que notre model aurait pu être plus proche de la réalité en ajoutant le principe de décision individuel pour avoir des chemins différents.

6 Réferences

— Social force model for pedestrian dynamics (Dirk Helbing et Péter Molnár)
— Lien vers le dépo git