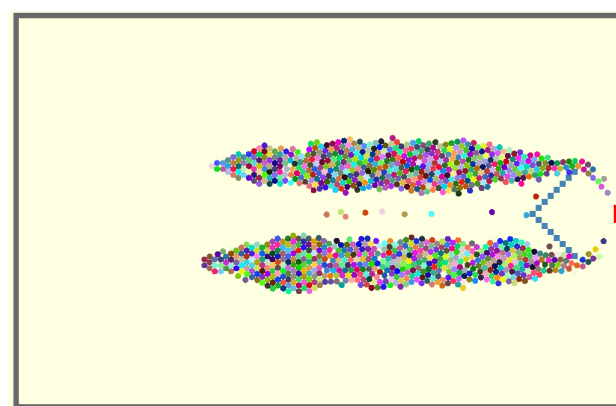
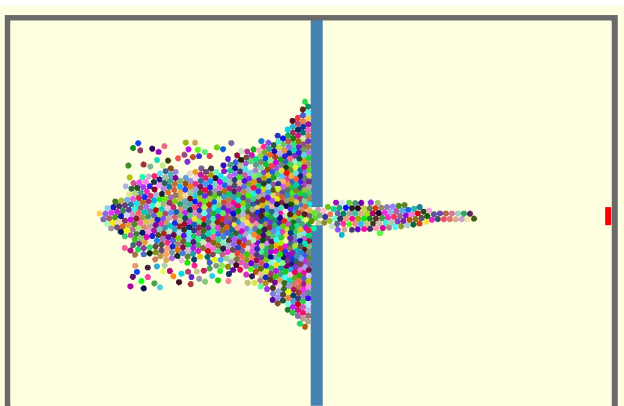
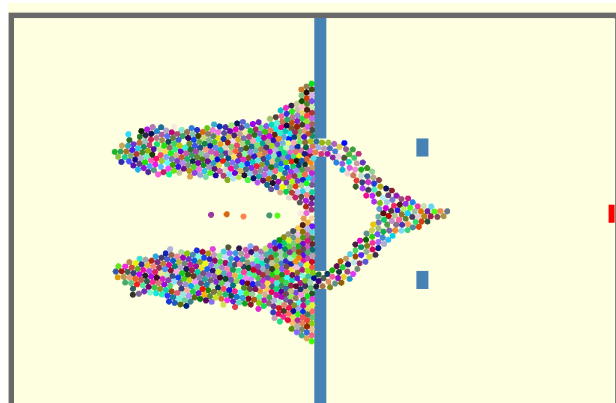
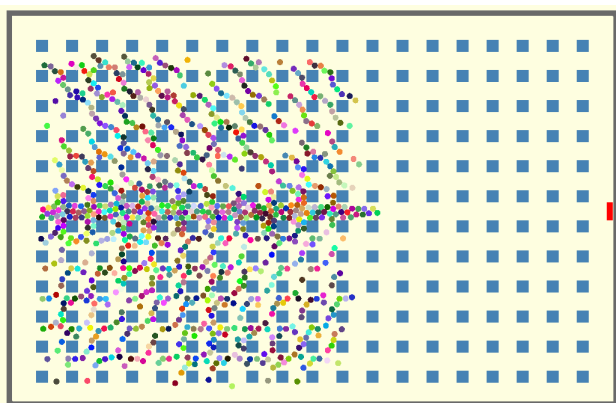


## PROJET C++ : MODÉLISATION NUMÉRIQUE DES MOUVEMENTS DE FOULE

---

Rudio Fida-Cyrille, Leo-Paul Baudet



# 1 Introduction

La Coupe du monde de football, ou Coupe du monde de la FIFA, est le championnat du monde des équipes nationales masculines de football créé le 28 mai 1928 par la Fédération internationale de football association (FIFA). C'est un événement très attendu de la part de tout les amateurs de foot. Néanmoins, l'édition 2022 qui a lieu au Qatar n'est pas sans faire polémiques : droits du travail, droits LGBT, droits des femmes...etc. Au vu du non respect de ces droits, l'organisation de la coupe du monde par le Qatar pose alors une question sur le respect des normes de sécurités dans les stades (au vu du non respects des droits fondamentaux). C'est pourquoi, afin de vérifier la bonne évacuation des stades en cas d'urgence et en fin de match, nous proposons une modélisation du mouvement de la foule. De plus, le choix de réaliser une modélisation est aussi en accord avec notre spécialité "Mathématiques Appliquées et Informatique" dont la simulation numérique est une des débouchés principales. Cela permet également de consolider nos acquis en analyse numérique, programmation et calcul haute performance.

## 2 Un bref historique


La modélisation mathématique et numérique des mouvements de foule est un modèle scientifique permettant d'étudier ou de simuler les déplacements d'un ensemble de personnes et/ou leur temps moyen d'évacuation d'une zone. Ce type de modélisation trouve son origine en 1987, lorsque Craig Reynolds a développé un programme informatique visant à simuler le comportement d'une nuée d'oiseaux en vol. Par ailleurs, ces dernières années, de plus en plus de chercheurs se sont intéressés à cette modélisation des mouvements de foule. En effet, la nécessité de simuler les mouvements d'un grand nombre de personnes se fait sentir, notamment lors de simulations d'évacuation, d'études de sécurité comme par exemple lors de la constructions de bâtiments ou encore lors de la réalisation de jeux vidéos

C'est pourquoi, dès 1997, les premières recherches furent effectuées dans le domaine de la simulation de foule, sous la direction de Daniel Thalmann, qui, avec l'aide de Soraia Raupp Musse, va établir une relation entre le comportement de l'individu seul au sein de la foule et le comportement résultant de l'interaction avec les autres individus de celle-ci. Trois ans plus tard, s'appuyant sur les travaux de Craig Reynolds, ces derniers vont développer le programme ViCrowd, un modèle de simulation créant en temps réel une foule virtuelle. Plus particulièrement, l'intérêt porté à ce genre de recherches s'est fortement accentué ces dernières années, notamment avec le contexte terroriste auquel nos sociétés sont confrontées, et qui pose alors la question de l'efficacité de l'évacuation des personnes se situant dans des espaces clos, ou encore plus récemment avec la COVID-19 et sa question de distanciation sociale entre individus. C'est ainsi que plusieurs modèles furent développés, certains associant la foule à un objet physique, et d'autres à un objet mathématique

Nous pouvons donc distinguer deux grands types de modélisation dépendant uniquement du facteur densité : une dite macroscopique, l'autre dite microscopique. La première modélisation est valable lorsque l'on étudie une densité très importante de personnes, et est basée sur une considération de la foule dans son ensemble (foule assimilée à un fluide, cadre de l'hydrodynamique utilisé), à l'inverse, lorsque la densité est plus faible, le second modèle permet lui de représenter chaque individu dans le temps et dans l'espace, en fonction notamment du comportement individuel de chacun, des décisions ou des interactions avec les autres. Lors de la modélisation microscopique, les individus sont assimilés à des particules (cadre de la dynamique newtonienne).

C'est ce dernier modèle que nous proposons de mettre en place de ce projet.

## 3 Présentation de l'application

Notre application permet alors de modéliser les mouvements de foule et plus spécifiquement l'évacuation des stades. Le programme est constitué d'une visualisation simple en 2D en temps réel de la simulation en cours, qui affiche les individus comme des sphères ainsi que la carte et les obstacles (voir ). Les calculs sont donc effectués en même temps et comme dans toute simulation, cela peut vite demander de grosses ressources lorsque la taille de la simulation augmente. La partie graphique de l'application est gérée par la bibliothèque graphique SDL2, une bibliothèque multimédia conçue pour le C, mais qui s'adapte bien à la programmation orientée objet. Notre programme permet à n'importe qui de lancer des simulations en modifiant les paramètres notamment les constantes liées aux différentes forces qui rentrent en jeu.

L'application contient également quelques "features" telles que la possibilité de se déplacer dans l'espace ou de zoomer/dezoomer pour mieux observer le comportement des individus. Également, comme pour toutes simulations, il est possible de modifier les paramètres de simulation assez facilement.

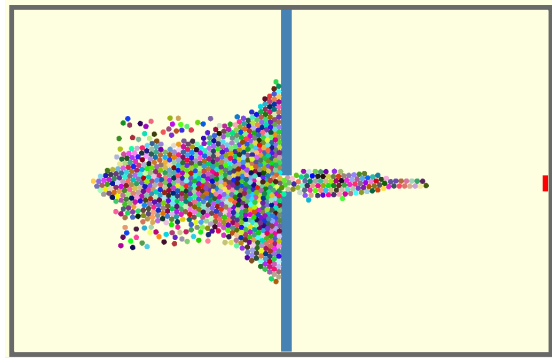


FIGURE 1 – Exemple de simulation

## 4 Modélisation

Le modèle que nous avons adopté est un modèle microscopique, celui des forces sociales.

Soit  $t$  la durée de l'expérience. On discrétise la durée de cette expérience en une subdivision égale  $(t_i)_{i \in \{0, \dots, n\}}$  tels que  $\forall i \in \{0, \dots, n\} \ t_i = t_0 + \frac{t}{n}$  où  $t_0$  est le temps du début de l'observation (de la simulation).

Il s'agit de créer des forces de répulsions (évitant aux piétons de s'interpénétrer et de rentrer dans les obstacles), une force d'accélération (permettant aux piétons de se diriger vers leur destination en fonction d'une certaine vitesse désirée) et une force de friction glissante (relatant le fait que lorsque deux piétons se télescopent, leur vitesse est modifiée). Ces forces seront calculées à chaque pas de temps pour chaque piéton.

A chaque pas de temps  $t_i$ , il s'agit ensuite de réaliser un PFD (deuxième loi de Newton) sur chaque piéton à partir des forces calculées précédemment. Enfin, il s'agit de résoudre l'équation différentielle obtenue à l'aide d'un schéma numérique afin d'obtenir la position de chaque piéton au cours de l'observation.

## 5 Modélisation de l'espace

### 5.1 Maillage et échelle

On représente l'espace de taille  $n \times m$  avec  $n \in \mathbb{N}$  et  $m \in \mathbb{N}$  par une matrice  $M \in \mathcal{M}_{(n+1) \times (m+1)}(\mathbb{R})$  (Les dimensions de la matrice seront expliquées après). Pour ne pas avoir une position uniquement décrite avec des coordonnées entières, on utilise une échelle afin d'avoir plus de précision : le décalage d'une colonne ou d'une ligne correspond à un déplacement  $p$  que nous appellerons pas de déplacement. Afin de délimiter l'espace, nous représentons les limites de celui-ci à l'aide de 1 sur les bords (d'où la dimension de la matrice). Les obstacles sont représentés par des 0. Voici un exemple de représentation de l'espace.

$$\begin{pmatrix} 1 & 1 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & \dots & 0 & 1 \\ 1 & 1 & \dots & 1 & 1 \end{pmatrix}$$

### 5.2 Obstacles

Les obstacles tels que les murs sont représentés exactement comme les limites de l'espace, à l'aide de 1. En effet, on assimile les bords de l'espace à des murs.

## 6 Forces prises en compte

Dans ce projet, nous prendrons en compte quatre forces comme citées plus haut.

## 6.1 Force d'accélération

C'est cette force qui va permettre de guider chaque individu vers le point où il souhaite aller, tout en évitant les obstacles. En effet, pour chaque piéton, on détermine un point d'arrivée (par exemple la sortie d'un stade). Voici l'expression de cette force :

$$\vec{F} = m * \frac{v_d * \vec{e}_a(t) - \vec{v}_i}{\tau_i} \quad (1)$$

Où  $m$  correspond à la masse du piéton,  $v_d$  à la vitesse désirée du piéton (fixée à 2m/s),  $\tau_i$  une constante de temps, appelée temps de relaxation, estimant le temps nécessaire au piéton pour retrouver sa vitesse désirée après un contact ou un changement soudain de direction,  $v_i$  la vitesse instantanée du piéton (à l'instant d'avant), et  $\vec{e}_a$  un vecteur que nous définissons ci après.

$\vec{e}_a$  est un vecteur donnant la direction vers le point d'arriver en évitant les obstacles, il change donc à chaque pas de temps.

$$\vec{e}_a = - \frac{\vec{\nabla}(\text{dist}(\vec{OM}))}{\|\vec{\nabla}(\text{dist}(\vec{OM}))\|} \quad (2)$$

La fonction distance donne la distance au point d'arrivée en fonction de la position, et  $\vec{OM}$  correspond au vecteur décrivant la position courante. L'idée est la suivante : la direction du gradient va des zones de champ faible vers les zones de champ fort (point d'arriver vers point de départ), nous souhaitons nous diriger des zones de champ fort vers les zones de champ de faible (d'où le - devant la force). On normalise le tout pour avoir la direction uniquement. Il nous reste donc à implémenter la fonction dist.

## 6.2 Retour sur l'espace : Algorithme de Dijkstra

Il s'agit ici d'implémenter la fonction dist. On peut voir cette fonction de la manière suivante avec  $x_a$  et  $y_a$  les coordonnées du point d'arrivée point d'arrivée :

$$f : \mathbb{R}^2 \rightarrow \mathbb{R} \quad (x, y) \mapsto \sqrt{(x - x_a)^2 + (y - y_a)^2} \quad (3)$$

Pour ce faire on, initialise chaque case de l'espace (autre que obstacles et bords) à une valeur infinie, et utilisons l'algorithme de Dijkstra pour calculer en chaque point de l'espace la valeur de  $f$ . Nous en utilisons une variante, l'algorithme de Fast-Marching.

L'idée est que l'on peut représenter chaque point de l'espace (discret) comme les sommets d'un graphe. Le but est alors de trouver le plus courts chemin d'un sommet vers un autre sommet. Voici la manière dont nous l'avons implémenté

Initialisation :

- On crée un tableau Distance de même taille que l'espace à visiter dont les différentes cases contiendront les distances au point de départ. Les différentes cases de ce tableau sont initialisées à l'infini.
  - On crée un tableau Fini de même taille que l'espace à visiter dont les différentes cases contiendront la valeur 1 si la case a déjà été traitée ou correspond à un obstacle et 0 sinon.
  - On crée une liste L contenant les couples de coordonnées des cases dont la distance a déjà été mise à jour au moins une fois mais dont on a pas encore regardé les voisins.
  - On ajoute la case de départ à la liste L et on met la case correspondante à 0 dans le tableau distance.
- Boucle (tant que la liste L n'est pas vide) :
- On cherche dans la liste L la case C dont la distance est la plus faible dans le tableau Distance.
  - On détermine les coordonnées des 4 voisins de la liste. Pour chaque voisin qui n'est pas marqué comme déjà traité dans le tableau Fini :
    - o On calcule la distance du voisin en utilisant la distance de la case C.
    - o Si celle-ci est inférieure à la valeur inscrite dans le tableau distance, on la met à jour.
    - o Si le voisin n'est pas dans la liste L, on l'ajoute.
  - On retire la case C de la liste L.
  - On marque la case L comme traitée dans le tableau Fini

### 6.3 Force répulsive d'interaction

La première force exercée entre deux piétons est la force répulsive d'interaction, qui a pour expression :

$$\vec{F} = A_i * \frac{r_{ab} - d_{ab}}{B_i} * \vec{u}_{ab} \quad (4)$$

avec  $r_{ab}$  la somme des rayons des piétons A et B,  $d_{ab}$  la somme des distances entre les deux piétons et  $\vec{u}_{ab}$  un vecteur unitaire du piéton A vers le piéton B,  $A_i$  et  $B_i$  deux constantes, où  $B_i$  est une distance caractéristique estimant l'ordre de grandeur à partir duquel la force deviendrait négligeable.

Cette force est une force à distance qui traduit la tendance psychologique de deux piétons à rester éloignés l'un de l'autre, en se tournant autour. Ainsi, elle est dirigée de telle sorte que le piéton a soit repoussé du piéton b. Contrairement à la force d'accélération si le piéton est suffisamment loin des autres piétons, cette force n'intervient pas (terme exponentielle négligeable) ;

### 6.4 Force de contact

Lorsque que les piétons sont à la limite de se toucher, ils subissent deux forces supplémentaires.

#### 6.4.1 Force du corps

Cette force a pour expression :

$$\vec{F} = k * (r_{ab} - d_{ab}) \vec{u}_{ab} \quad (5)$$

avec k une constante.

Comme la force de répulsion, cette force est dirigée de telle sorte que le piéton a soit repoussé par le piéton b.

On remarque que cette force à la forme d'une force de rappel, dont la constante de rappel k est très grande. Ainsi, les piétons ont beaucoup de difficultés à se rapprocher l'un de l'autre. À noter qu'il est nécessaire que cette constante soit très grande. En effet, si elle était trop faible, les piétons rentreraient, malgré cette force, dans les autres piétons ou dans les murs. De plus, cette force est conservative. En effet, si les piétons se touchent, ils ne perdent pas leur vitesse mais rebondissent l'un sur l'autre.

#### 6.4.2 Force de friction glissante

Son expression est la suivante :

$$\vec{F} = K * (r_{ab} - d_{ab}) (v_a(t) - v_b(t)) \vec{t}_{ab} \quad (6)$$

avec  $\vec{t}_{ab}$  un vecteur tangent à la trajectoire,  $v_a$  et  $v_b$  la vitesse du piéton a et b, K une constante.

Il s'agit d'une force de frottement due à une différence de vitesse entre les deux piétons. Cette force, qui est tangente au piéton a considéré, traduit le fait que si deux piétons se touchent, ils peuvent glisser l'un sur l'autre et s'entraîner.

Par exemple, ici, si les piétons a et b se déplacent tous les deux vers le haut mais que le piéton b possède une vitesse plus faible que le piéton a, alors le piéton a va glisser sur le piéton b, contraignant le piéton a à ralentir et provoquant alors une force de frottement dirigée vers le bas. À l'inverse, si le piéton b a une vitesse plus grande, alors le piéton b va entraîner le piéton a et la force de frottement sera, cette fois-ci, vers le haut.

## 7 Schéma numérique choisi

Pour calculer la force d'accélération, il reste à calculer le gradient le gradient de la fonction dist. Pour cela nous utilisons le schéma des différences finies pour approximer les dérivées partielles :

$$\vec{\nabla}(\text{dist}(x, y)) = \left( \frac{\partial \text{dist}}{\partial x}(x, y), \frac{\partial \text{dist}}{\partial y}(x, y) \right) \quad (7)$$

On estime alors ses composantes par :

$$\frac{\partial \text{dist}}{\partial x}(x, y) = \text{dist}(x + 1, y) - \text{dist}(x - 1, y) \quad (8)$$

$$\frac{\partial \text{dist}}{\partial x}(x, y) = \text{dist}(x, y + 1) - \text{dist}(x, y - 1) \quad (9)$$

Enfin, une fois toutes les forces calculées, il reste à résoudre l'équation différentielle donnée par le PFD, afin d'obtenir d'une part la vitesse puis la position. Ainsi, nous avons implémenter un schéma d'euler explicite pour résoudre cette équation.

A l'aide du PFD nous obtenons l'équation différentielle suivante, pour le piéton numéro  $i$  à l'instant  $t$  :

$$\sum \vec{F}^i(t) = m * \vec{a}^i(t) \quad (10)$$

Où  $\sum \vec{F}^i(t)$  correspond à la résultante des forces extérieur s'appliquant sur le piéton numéro  $i$  à l'instant  $t$  et  $\vec{a}^i(t)$  l'accélération du piéton  $i$  à l'instant  $t$ .

En projetant selon  $x$  et  $y$ , nous obtenons les équation suivantes :

$$\begin{cases} \sum F_x^i(t) &= m * a_x^i(t) \\ \sum F_y^i(t) &= m * a_y^i(t) \end{cases} \quad (11)$$

Où  $\sum F_x^i(t)$  la résultante des forces selon  $x$  du piéton  $i$  à l'instant  $t$ ,  $\sum F_y^i(t)$  la résultante des forces selon  $y$  du piéton  $i$  à l'instant  $t$ ,  $a_x^i(t)$  l'accélération du piéton  $i$  selon  $x$  à l'instant  $t$  et  $a_y^i(t)$  l'accélération du piéton  $i$  selon  $y$  à l'instant  $t$

En réa-rangeant les termes, on obtiens l'équation suivante :

$$\begin{cases} \frac{dv_x^i(t)}{dt} &= f_x(t, v_x^i(t)) \\ \frac{dv_y^i(t)}{dt} &= f_y(t, v_y^i(t)) \end{cases} \quad (12)$$

Avec :

$$\begin{aligned} \mathbb{R}^2 &\rightarrow \mathbb{R} \\ f_x : (t, y(t)) &\mapsto \frac{\sum F_x^i(t)}{m} \end{aligned} \quad (13)$$

et

$$\begin{aligned} \mathbb{R}^2 &\rightarrow \mathbb{R} \\ f_y : (t, y(t)) &\mapsto \frac{\sum F_y^i(t)}{m} \end{aligned} \quad (14)$$

On peut donc résoudre ces équations différentielles à l'aide d'un schéma numérique explicite, à chaque instant  $t_n$  de la discrétisation temporelle. Notons  $h = t_{k+1} - t_k$  le pas de discrétisation temporelle ( $t_k = i * \frac{t}{k}$ ) :

$$\begin{cases} v_{x,n+1}^i(t_n) &= v_{x,n}^i(t_k) + h * f_x(t_k, v_{x,n}^i(t_k)) \\ v_{x,0}^i(t_n) &\in \mathbb{R} \end{cases} \quad (15)$$

$$\begin{cases} v_{y,n+1}^i(t_k) &= v_{y,n}^i(t_k) + h * f_y(t_k, v_{y,n}^i(t_k)) \\ v_{y,0}^i(t_k) &\in \mathbb{R} \end{cases} \quad (16)$$

Après un certain nombre d'itération  $n_0$  on obtient,  $v_{y,n_0}^i(t_k) \simeq v_y^i(t_k)$  et  $v_{x,n_0}^i(t_k) \simeq v_x^i(t_k)$

A partir de ces vitesses, on détermine la position de chaque piéton à l'aide d'un autre schéma d'euler, on a :

$$\begin{cases} \frac{dx^i(t)}{dt} &= v_x^i(t) \\ \frac{dy^i(t)}{dt} &= v_y^i(t) \end{cases} \quad (17)$$

$$\begin{cases} x_{n+1}^i(t_k) &= x_n^i(t_k) + h * v_x^i(t_k) \\ y_{n+1}^i(t_k) &= y_n^i(t_k) + h * v_y^i(t_k) \end{cases} \quad (18)$$

On obtient de même au bout d'un certain nombre d'itération  $n_1$ ;  $x_{n_1}^i(t_k) \simeq x^i(t_k)$  et  $y_{n_1}^i(t_k) \simeq y^i(t_k)$

## 8 Implémentation en C++

Le C++ est langage très utilisé pour effectuer des modélisations notamment de problème physique ou mécanique comme dans notre cas. En effet, étant donné qu'il est bas niveau, il offre de très bonnes performances qui permettent d'effectuer des opérations assez lourdes. De plus, la programmation orientée objet permet de modéliser de manière concrète chaque élément de la simulation ainsi que son comportement, ce qui simplifie grandement l'implémentation. On pourra également, noter la possibilité de mettre en place des pratiques de HPC pour booster nos performances, comme par exemple la parallélisation multi-thread avec OpenMP qui se met facilement en place.

## 8.1 Classes et diagramme UML

L'organisation de notre programme et des classes qui sont présentées schématiquement dans la figure 2

### Simulation

La classe principale du programme est la classe **Simulation** qui regroupe toutes la modélisation de l'espace dans une instance **Espace** ainsi que de la foule dans une instance **ModeleFoule**. Il y a également un pointeur vers un objet **Fast-Marching** qui est utilisé pour calculer la force d'accélération des individus.

### Affichage

La quasi totalité de la partie graphique, utilisant la SDL2, est gérée dans la classe abstraite **GraphicWindow** dont hérite **Simulation**. Cela permet une gestion simplifiée et compacte de l'affichage de la modélisation. Cette approche permet d'utiliser efficacement les fonctions et structures de données C de la SDL2 avec la programmation orientée objet.

### Mécanique

D'un point de vue mécanique, l'utilisation d'une classe **Vec2D** permet de simplifier les opérations effectuées sur les vecteurs en 2 dimensions. Cela se fait notamment à travers l'utilisation de surcharges d'opérateur (qui ne font pas partie de la classe). La classe **Vec2D** est un template ce qui permet de réutiliser les vecteurs pour stocker des coordonnées entières.

### Modélisation des individus

Les individus/piétons sont représentés par la classe abstraite **Individu**. Cette classe est héritée notamment par la classe **IndividuDisque** qui représente un individu comme étant un disque. La structure du programme permet de profiter le polymorphisme et donc de créer de nouvelles formes d'individu selon les besoin.

### Modélisation de la foule

Dans la classe **ModeleFoule**, la population d'individus est stockée dans une instance de **vector** de pointeurs. Les vecteurs sont intéressants dans notre cas étant donné que la taille de la population est fixe pendant la majorité de la simulation et que l'on itère de nombreuses fois sur les individus de la population. L'utilisation de pointeurs et non d'instance d'individu permet de bénéficier du polymorphisme : de nouvelles formes d'individu peuvent être implémentées sans nécessiter de gros changements dans le code actuel.

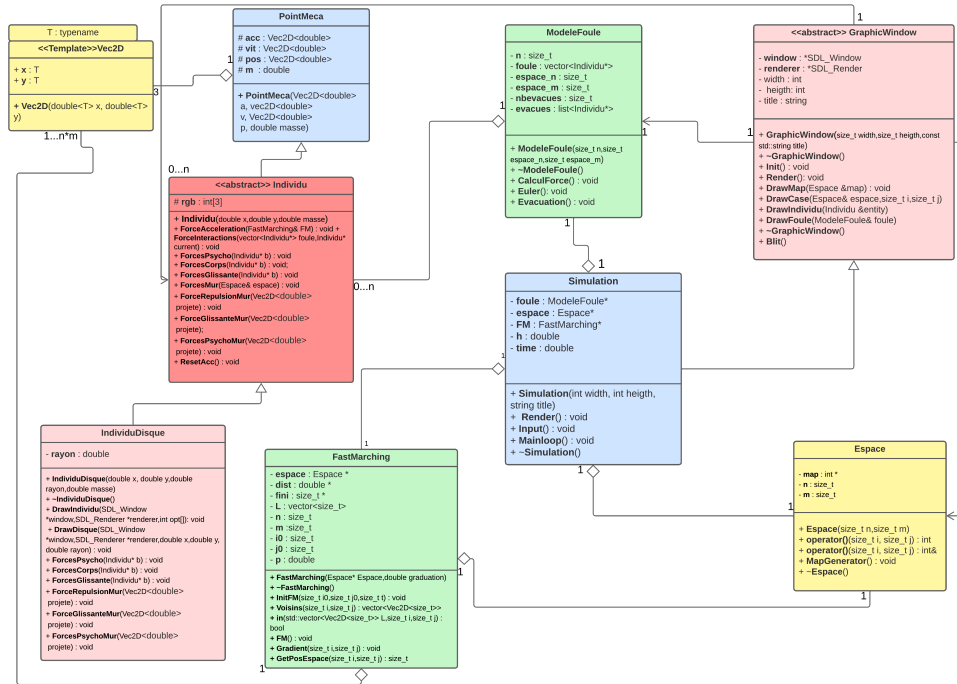


FIGURE 2 – Diagramme UML de l'application

## 8.2 Optimisation avec OpenMP

Les calculs effectués pour la simulation ont une complexité élevée. En effet, la plupart sont en  $\mathcal{O}(N^2)$ ,  $N$  étant le nombre d'individu. Cela ralentit alors grandement la simulation lorsque le nombre d'individu est élevé. Les boucles principales de calcul, dans `CalculForce` et `Euler` de `ModeleFoule.hpp` ont été parallélisées avec `OpenMp` pour augmenter les performances.

## 9 Installation et utilisation

Pour rappel, les procédures d'installation sont également données dans le Readme de notre [Repository Github](#).

### 9.1 Installation

Le programme dépend de trois bibliothèques différentes : la SDL2, SDL-ttf et OpenMP.

Voici les commandes sous Ubuntu pour les installer.

```
sudo apt install libsdl2-dev libsdl2-ttf-dev libomp-dev
```

Pour compiler et lancer le programme :

```
make
./foule
```

Pour lancer les test :

```
make test
```

### 9.2 Utilisation

Les paramètres de la simulation sont accessibles dans le fichier "parametres.hpp". Cela permet d'avoir des simulations sur-mesure et de tester plusieurs type de simulations selon ce que l'on veut observer. Les paramètres de base du programme permettent d'avoir des simulations réalistes. Il est également possible de se déplacer sur l'affichage en utilisant les **flèches**, de zoomer avec la touche **K** et de dézoomer avec **J**.

## 10 Fierté

Parmi les parties du code dont nous sommes les plus fiers, nous pouvons citer la gestion de l'affichage qui est "scalable", ce qui permet les déplacements ainsi que les zooms/dezooms, l'implémentation du Fast-Marching que nous apprécions pour la simplicité et la praticité de l'algorithme. L'utilisation d'une classe pour les vecteurs à 2 dimensions est simple et logique mais nous rend particulièrement fiers étant donné qu'elle permet de simplifier le code qui se rapproche alors d'API comme **Numpy** en Python. Finalement, nous sommes également fiers d'avoir pu intégrer des connaissances en HPC avec OpenMP.

## 11 Conclusion

Ce projet nous a tout d'abord permis de mettre en oeuvre une grande partie des notions que nous avons vu en MAIN jusqu'à présent : analyse numérique(euler), parallélisation (openmp), algorithmique (bfs sur un graphe) et programmation orienté objet en C++ (le but premier du projet). De plus, nous avons établi notre propre modèle mathématique pour cette simulation. Ces deux éléments nous rendent fier de ce projet.

L'application développée remplit parfaitement son rôle et est donc satisfaisante en ce sens. La partie graphique est également assez dynamique et agréable. Cependant, l'application peut encore être largement améliorée, notamment au niveau du rendu de texte sur la fenêtre, et des méthodes de calcul de certaines forces qui pourraient être optimisées. De plus, une meilleure gestion des paramètres peut être implémentée, pour éviter la recompilation.

Voici pour finir quelques exemples de simulation que nous pouvons obtenir à l'aide de notre application :

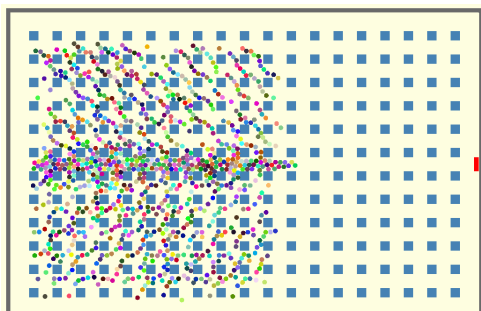


FIGURE 3 – Configuration de "Jussieu"

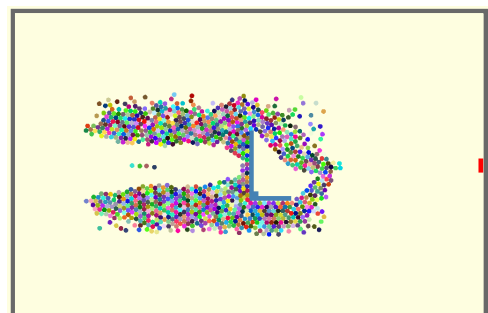


FIGURE 4 – Configuration avec l'initiale (la plus simple a dessiner) d'un des membres du projet