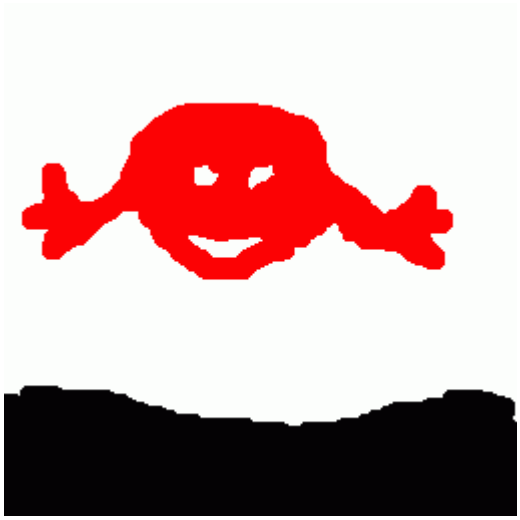
# Projet
# Maths pour l'info
## - IMAC 1 -

# Système masse – ressort



Billard Julie
Fayolle Séverine

# INTRODUCTION

Le but premier de ce projet est de réaliser une animation d'un carré de gélatine lors d'une chute. Cette gélatine devra rebondir sur le sol.

Pour se faire le programme devra à partir d'un fichier de configuration et d'une scène initiale au format « ppm » générer des images, elles aussi au format « ppm », après avoir fait les calculs permettant une modification des images pour chaque itération.

La définition de ces calculs sera obtenue par des formules mathématiques et physiques notamment sur le calcul des forces et sur la relation fondamentale de la dynamique.

Pour suivre cette évolution, ce rapport s'articule en deux axes : une partie mathématiques utilisant des formules physiques et une partie programmation.

# PARTIE 1 : MATHEMATIQUES ET PHYSIQUES

## A- Recherche des formules et de leurs applications.

Nous savons comment calculer les forces, et par conséquent, comment calculer l'accélération. Or la vitesse d'une particule à un temps t correspond à son ancienne vitesse à laquelle on ajoute l'accélération de la particule sur une période DT (DT étant l'intervalle de temps entre chaque nouveau calcul de la vitesse).

$$V(t) = V(t-1) + A(t) * DT$$

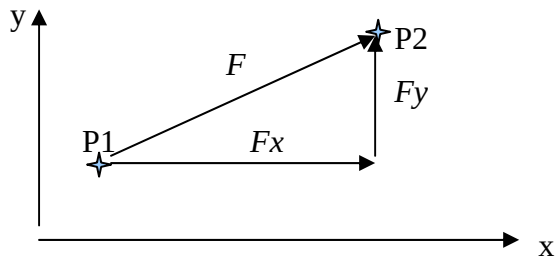
Afin de faciliter les calculs, les forces, accélérations et vitesses seront réparties selon l'axe des abscisses et selon l'axe des ordonnées dans un plan orthogonal.

A partir de la vitesse, on déduit la distance parcourue pendant la période DT. Il suffit alors d'ajouter cette distance à l'ancienne position de la particule pour trouver sa nouvelle position.

$$Pos_x(t) = Pos_x(t-1) + V_x(t) * DT$$
$$Pos_y(t) = Pos_y(t-1) + V_y(t) * DT$$

## B- Orientation des forces.

Le problème qui se pose maintenant est de savoir comment une force se répartie selon l'axe des abscisses et l'axe des ordonnées, puisque la formule données dans le sujet n'en tient pas compte.




Une force ***F*** peut être exprimées selon ces deux axes tel que : $\vec{F} = \vec{F}_x + \vec{F}_y$

Pour calculer la force exercée par un ressort, nous avons besoin de connaître la distance au repos de ce ressort. Nous distinguerons donc deux distances au repos pour le ressort :

- Une selon l'axe des abscisses.
- Une autre selon l'axe des ordonnées.

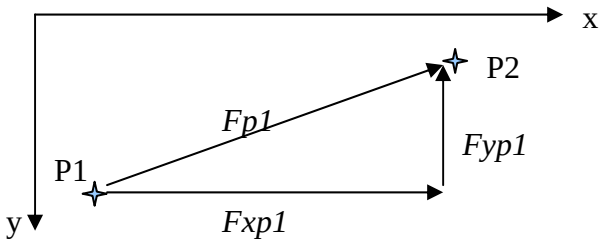
Ainsi nous n'avons pas de calcul à effectuer pour passer de la force ***F*** aux forces ***Fx*** et ***Fy*** qui sont nécessaires pour obtenir les nouvelles positions des particules.

Il faut maintenant savoir dans quel sens s'exerce les forces sur **P1** et **P2** afin d'en faire une somme correcte.
Dans notre cas, les forces recherchées sont celles exercées par un ressort, on a donc :

- Attraction lorsque la distance entre les deux particules est supérieure à la distance au repos du ressort.
- Répulsion lorsque la distance entre les deux particules est inférieure à la distance au repos du ressort.

De plus, avec la formule donnée dans le sujet, la force exercée par le ressort est positive en cas d'attraction et négative en cas de répulsion.

<u>Considérons deux particules dans les cas de l'attraction :</u>




Dans le cas de l'attraction, les forces ***Fx*** et ***Fy*** sont positives.

$P1_x > P2_x$ donc $F_{xp1} < 0$ et $\vec{F}_{xp1} = \vec{F}_x$

$P1_y < P2_y$ donc $F_{yp1} > 0$ et $\vec{F}_{yp1} = -\vec{F}_y$

Donc on obtient :

$P1_x < P2_x \rightarrow \vec{F}_{xp1} = \vec{F}_x$

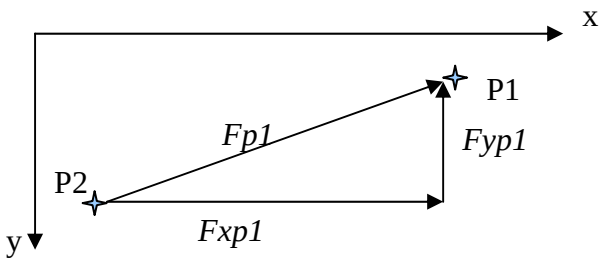
$P1_x > P2_x \rightarrow \vec{F}_{xp1} = -\vec{F}_x$

$P1_y < P2_y \rightarrow \vec{F}_{yp1} = \vec{F}_y$

$P1_y > P2_y \rightarrow \vec{F}_{yp1} = -\vec{F}_y$

Vérifions que ce résultat est toujours valable dans le cas d'une force de répulsion.

Considérons deux particules dans les cas de la répulsion :




Dans le cas de la répulsion, les forces ***Fx*** et ***Fy*** sont négatives.

$P1_y$ p $P2_y$ donc $F_{yp1}$ p 0 et ←

$P1_x$ f $P2_x$ donc $F_{xp1}$ f 0 et $F_{xp1} = -F_x$

Donc on obtient :

$P1_x$ f $P2_x$ ➔ $\vec{F}_{xp1} = \vec{F}_x$

$P1_x$ p $P2_x$ ➔ $\vec{F}_{xp1} = -\vec{F}_x$

$P1_y$ f $P2_y$ ➔ $\vec{F}_{yp1} = \vec{F}_y$

$P1_y$ p $P2_y$ ➔ $\vec{F}_{yp1} = -\vec{F}_y$

Afin d'additionner correctement les différentes forces s'exerçant sur une particule, nous compareront donc les positions des particules.

# PARTIE 2 : PROGRAMMATION

## A- Présentation de l'algorithme général.

---

But de cet algorithme : générer un certain nombre d'images représentant la chute d'un objet gélatineux et ses interactions avec le décors.
**Paramètres : <nom du fichier >.conf  <nom du fichier> .ppm**

---

**Début**

Entier : i ← 0
Entier : iterationImage

- Ouverture du fichier .conf et récupération des données *(cf : II-B-1)*
- Ouverture du fichier .ppm et récupération du format et des couleurs des pixels *(cf : II-B-1)*
- Création et initialisation du tableau de particules *(cf : II-B-2)*
- Création et initialisation du tableau de ressorts *(cf : II-B-3)*

iterationImage = NB_ITERATIONS / NB_IMAGES
Tant que i < NB_ITERATIONS souhaité dans le fichier .conf
- Calcul du mouvement de chaque particule et mise à jour *(cf : I)*
  Si i est un multiple de iterationImage
  - Trier les particules dans leur nouvel ordre *(cf : II-B-4)*
  - Generer l'image *(cf : II-B-1)*

  FinSi
- Incrémenter i

FinTantque

**Fin**

## B- Algorithmes détaillés.

### 1) Gestion de fichiers et création d'images PPM.

---

Fonction **lectureConf** (chaîne de caractères nomFichier) retourne entier.
*//cette fonction permet la récupération des données du fichiers .conf, elle retourne 1 si tout se passe bien, -1 sinon.*

---

**Début**

- Ouverture du fichier nomFichier

Si l'ouverture a échoué Retourner -1 FinSi

- Initialisation de la variable globale DT avec la 2ème ligne du fichier.
- Initialisation de la variable globale GRAVITE avec la 4ème ligne du fichier.
- Initialisation de la variable globale MASSE avec la 6ème ligne du fichier.
- Initialisation de la variable globale TENSION_RESSORT avec la 8ème ligne du fichier.
- Initialisation de la variable globale NB_ITERATIONS avec la 10ème ligne du fichier.
- Initialisation de la variable globale NB_IMAGES avec la 12ème ligne du fichier.

- Fermeture du fichier

Si la fermeture a échoué Retourner -1 FinSi

- Retourner 1

**Fin**

---

Fonction **lecturePPM** (chaîne de caractères nomFichier) retourne chaîne de caractères.
*//fonction permettant la récupération du contenu d'un fichier .ppm*

---

**Début**

Chaîne de caractères : Image

- Ouverture du fichier nomFichier

Si l'ouverture a échoué Retourner NULL FinSi

- Récupération du contenu de la première ligne du fichier (= mode de couleur du fichier)

Si le mode de couleur n'est pas « P6 » Retourner NULL FinSi

- On passe toutes les lignes commençant par # (ligne de commentaires).
- Initialisation de la variable globale HAUTEUR avec le 1er entier contenu dans la 1ère ligne suivant les commentaires.
- Initialisation de la variable globale LARGEUR avec le 2ème entier contenu dans la 1ère ligne suivant les commentaires.
- On passe la ligne suivante.

- Allocation d'un espace mémoire de taille HAUTEUR*LARGEUR pour la variable Image.
- Image ← l'ensemble des caractères modélisant la couleur de chaque pixel de l'image.ppm ouverte.

- Fermeture du fichier.

Si la fermeture a échoué Retourner NULL FinSi

- Retourner Image.

**Fin**

Fonction **ecriturePPM** (chaîne de caractères nomFichier) retourne entier
*//fonction permettant l'écriture de l'entête du fichier .ppm*

**Début**

- Ouverture du fichier nomFichier.

Si l'ouverture a échoué Retourner -1 FinSi

- Ecrire sur la 1ère ligne « P6\n ».
- Ecrire sur la 2nde ligne le format de l'image : HAUTEUR, LARGEUR.
- Ecrire sur la 3ème ligne « 255\n».
- Fermeture du fichier.

Si la fermeture a échoué Retourner -1 FinSi

- Retourner 1.

**Fin**

Fonction **ecritureCouleur** (chaîne de caractères nomFichier, tableau de pointeurs sur particule Image) retourne entier
*//fonction permettant l'écriture à la suite du fichier .ppm des couleurs de ses pixels, le tableau Image pointe les particules dans leur nouvel ordre*

**Début**

Entier : i

- Ouverture du fichier nomFichier

Si l'ouverture a échoué Retourner -1 FinSi

Pour (i ← 0, i<HAUTEUR*LARGEUR, i++)
Si image[i] n'est pas vide

- Ecrire à la suite du fichier image[i]→couleur[0]
- Ecrire à la suite du fichier image[i]→couleur[1]
- Ecrire à la suite du fichier image[i]→couleur[2]

Sinon
*//Les particules en bougeant ont laissé un espace vide qui devra être considéré comme de l'air donc comme étant blanc.*

- Ecrire à la suite du fichier le caractère de code ASCI 255
- Ecrire à la suite du fichier le caractère de code ASCI 255
- Ecrire à la suite du fichier le caractère de code ASCI 255

FinSi
FinPour

- Fermeture du fichier

Si la fermeture a échoué Retourner -1 FinSi

- Retourner 1

**Fin**

## 2) Initialisation des particules.

---

Procédure **creerParticules** (tableau de particules Image, chaîne de caractères couleur)
*//chaque case du tableau couleur contient une valeur. Si on prend trois cases par trois cases, on obtient les valeurs RVB de la couleur d'une particule.*

---

**Début**
Entier : i

Pour (i ← 0, i < HAUTEUR*LARGEUR, i++)
- Initialiser à zéro forces, accélérations, vitesses de Image[i]

- Image[i].couleur[0] ←couleur[3*i]
- Image[i].couleur[1] ←couleur[3*i+1]
- Image[i].couleur[2] ←couleur[3*i+2]

- Image[i].positionX ←i % LARGEUR
- Image[i].positionY ←i / LARGEUR

Si Image[i] est une particule rouge
- Image[i].masse ← MASSE

Sinon
*//on prend une masse qui annulera la force de gravitée au moment du calcul des forces évitant ainsi de tester à chaque fois s'il s'agit d'une particule rouge.*
- Image[i].masse <- 0

FinSi
FinPour
**Fin**

## 3) Initialisation des ressorts.

Lors de l'allocation de l'espace mémoire du tableau de ressorts, on prend comme taille le nombre maximum de ressort que peut posséder l'image, c'est-à-dire le cas où toute les particules possèdent un ressort. Les ressorts inutiles seront donc initialisés de façon à ce qu'ils n'exercent aucune force.

---

Procédure **initialiserRessort** (ressort Ressort)
*//cette procédure calcule la distance repos en x et la distance repos en y pour les ressorts reliant deux particules rouges, les autres ressorts auront des distances au repos nulles. Elle attribue également la tension ressort du fichier .conf aux ressorts reliant deux particules rouges et une tension nulle pour les autres. Ainsi, les particules blanches et noires possèdent également des ressorts mais ceux-ci ont une tension nulle et sont donc comme inexistants.*

---

Procédure **creerRessorts** (tableau de ressorts Ressort, tableau de particules Image)
*//procédure permettant de créer les ressorts entre les différentes particules de l'image et d'initialiser leur valeur.*

**Début**

Entier : i
Entier : j ← 0

Pour (i ←0, i < HAUTEUR*LARGEUR, i++)
Si Image[i] possède un voisin de dessus
- Ressort[j].particule1 pointe sur Image[i]
- Ressort[j].particule2 pointe sur Image [i-LARGEUR]
- Initialiser le ressort à l'aide de la procédure initialiserRessort
- j++ *//on passe au ressort suivant*

FinSi

Si Image[i] possède un voisin de droite
- Ressort[j].particule1 pointe sur Image[i]
- Ressort[j].particule2 pointe sur Image[i+1]
- Initialiser le ressort à l'aide de la procédure initialiserRessort
- j++

FinSi

Si Image[i] possède un voisin en haut à droite
- Ressort[j].particule1 pointe sur Image[i]
- Ressort[j].particule2 pointe sur Image[i+1-LARGEUR]
- Initialiser le ressort à l'aide de la procédure initialiserRessort
- j++

FinSi

Si Image[i] possède un voisin en bas à droite
- Ressort[j].particule1 pointe sur Image[i]
- Ressort[j].particule2 pointe sur Image[i+1+LARGEUR]
- Initialiser le ressort à l'aide de la procédure initialiserRessort
- j++

FinSi
FinPour

**Fin**

## 4) Algorithme de tris des particules.

Pour trier les particules dans leur nouvel ordre au moment de la génération d'image, nous avons créé un tableau de pointeurs sur particules de la taille de l'image. Ainsi, lorsqu'une particule changera de position, elle ne bougera pas dans le tableau de particules grâce auquel on effectue tous les calculs, mais dans le tableau de pointeurs.
Par exemple, une particule située à la case (2,3) du tableau de particule a donc comme position d'origine dans l'image la deuxième ligne, troisième colonne. Si elle bouge à la ligne du dessous, on ne peut pas la replacer dans le tableau de particules à la case (3,3) car on risquerait d'écraser et de perdre les données d'une autre particule qui seraient également arrivée à la case (3,3). On se sert donc du tableau de pointeurs. Le pointeur de la case (3,3) de ce tableau pointera sur la particule ayant bougée en (3,3). Les particules rouges sont prioritaires sur les blanches. Ainsi lorsque l'on parcourra le tableau de pointeurs, on retrouvera bien les particules dans leur nouvel ordre, et on pourra donc écrire dans le bon ordre les couleurs des particules lors de la génération d'image. Si un pointeur est NULL au moment de l'écriture dans le fichier .ppm, on considérera qu'il s'agit du vide laissé par le déplacement des particules et on écrira donc la couleur blanche.
Avant chaque nouveau tris, les pointeurs du tableau sont réinitialisés à NULL.

# PARTIE 3 : BILAN DU TRAVAIL

## A- Problème rencontré et solution apportée.

Le principal problème que nous avons eu se trouve dans le calcul des forces. Notre premier calcul se décomposait en calcul de la force résultante, puis en projection de cette force selon l'axe des abscisses et l'axe des ordonnées.

Nous avons simplifié les calculs en prenant le projeté de la force en fonction de x sur l'axe (O,x) et en faisant de même pour l'axe (O,y). Ainsi les distances au repos entre chaque particule sont toujours égales à 1 ou 0.

Nous avons aussi du modifier la structure des ressorts afin de pouvoir appliquer cette méthode en remplaçant :

```
Double distanceRepos ;
```

Par :

```
Double distanceReposX ;
Double distanceReposY ;
```

## B- Travail réalisé.

Notre programme implémenté en langage C permet de lire un fichier de configuration et une scène entrés sur la ligne de commande et de générer une série d'images représentant la scène à différents moments.
Il est composé de fichiers .C et .H et il se compile grâce à un Makefile. Il comporte aussi un fichier d'aide.

Nous n'avons par contre pas eu le temps de rajouter des options telles que le vent, les couleurs ou les collisions.