Osmos

laurent.jospin.59@free.fr, http://jospin.lstl.fr

Lycée Saint-Louis, Paris

Ce sujet s'inspire d'un jeu développé par Hemisphere Games et publié sous le nom de Osmos.

Ce jeu utilise des entités constituées de matière ou d'anti-matière qui évolue dans un espace en deux dimensions. Lorsque deux entités de même nature se rencontrent, l'une absorbe l'autre avec un taux de transfert qui dépend de la taille de l'interface qui les relie. Lorsque deux entités de nature différente se rencontrent, les deux s'amputent mutuellement avec le même taux de transfert que dans le cas précédent.

Les entités peuvent se déplacer en "dégazant", c'est-à-dire en se séparant d'une fraction de leur masse à l'un des points de leur contour. La partie dégazée forme une nouvelle entité et les deux entités se retrouvent propulsées en sens inverse l'une de l'autre. La raison d'être des entités est de grossir au maximum.

On représente dans le sujet une entité par un quintuplet *(position, vitesse, rayon, nature, joueur)* où *position* et *vitesse* sont eux-mêmes des couples de coordonnées (dans le programme, on utilisera dans les deux cas des listes plutôt que des tuples). Les entités se déplacent dans un espace clos correspondant au carré unitaire $OIKJ$ dans un repère orthonormé (O,I,J) avec $K(1,1)$.

Les entités ont une masse qui est égale au carré de leur rayon.

1 Un peu de géométrie

Dans le programme comme dans le sujet, on assimile un vecteur à ses coordonnées, de même pour un point.

1. Ecrire une fonction `vecteur` qui prend deux points en argument et qui renvoie le vecteur qui les relie.

2. Ecrire une fonction `ajoute` qui prend un point et un vecteur en argument, et qui renvoie le translaté du point par le vecteur.

3. Ecrire une fonction `norme` qui prend un vecteur en argument et renvoie sa norme.

2 Initialisation de l'univers

On pourra utiliser le module `random` notamment la fonction `random()` qui renvoie un flottant de l'intervalle `[0,1[` et la fonction `choice` qui choisit un élément au hasard dans une liste.

4. Ecrire des fonctions `position`, `vitesse`, `rayon`, `nature`, `joueur` et `masse` prenant en argument une entité et renvoyant la propriété correspondante.

5. Ecrire une fonction `collision(entite1, entite2)` qui renvoie si deux entités sont en collision en comparant leur distance par rapport aux rayons des entités.

6. Ecrire une fonction `initialise_entites(entites, n, rayonMax)` qui prend une liste d'entités potentiellement non vide et qui la complète jusqu'à ce qu'elle contienne précisément n entités en y ajoutant des entités qui n'entrent pas en collision ayant un rayon compris dans $[\frac{rayonMax}{10}, rayonMax[$, une vitesse nulle, une nature choisie aléatoirement entre `MATIERE` et `ANTIMATIERE` (déjà définies en début de fichier) et un joueur égal à `None`.

7. Définir également une fonction `couleur` renvoyant la couleur d'une entité. On renverra `"green"` si l'entité appartient à un joueur, `"black"` si l'entité est constitué d'anti-matière et `"white"` sinon.

3 Représentation de l'univers

8. La méthode `create_oval` des `Canvas` permet de représenter un oval (et donc en particulier aussi un cercle) en lui donnant en argument les coordonnées extrêmes de l'oval x_{min}, y_{min}, x_{max} et y_{max}. On peut choisir la couleur de remplissage grâce à l'argument non positionnel `fill`. Le `Canvas` porte ici le nom de `surfaceDeDessin` et l'instruction : `surfaceDeDessin.create_oval(20, 40, 30, 50, fill="blue")` permet ainsi de tracer un cercle de diamètre 10 pixels, de centre (25, 45) rempli de la couleur bleue.

Compléter votre programme en ajoutant à la fonction `affichage` le tracé de toutes les entités. Attention, les coordonnées des entités sont dans [0,1[tandis que dans la fenêtre graphique ils sont en pixels, il faut donc multiplier les coordonnées par la dimension de la surface de dessin `DIMENSION`. Décommenter l'initialisation dans `cree_fenetre` et tester l'affichage obtenu.

4 Mise en mouvement

9. Ecrire une fonction `deplacement(entites)` qui modifie chaque entité de la liste en argument en translatant l'entité de vitesse $\vec{v}$ du vecteur `VITESSE` $\times \vec{v}$. En procédant ainsi, l'entité peut sortir du carré ce que l'on corrigera ainsi : si l'entité se retrouve à déborder sur un bord, on déplacera son centre de sorte que ce ne soit plus le cas (on la décale contre le bord), et on changera le signe de la composante de la vitesse correspondant au rebond.

Dans la fonction `pas_de_jeu`, appeler `deplacement(entites)`.

10. Ecrire une fonction `masse_de_rayon(rayon)` et `rayon_de_masse(masse)` qui donne la masse pour un rayon donné et réciproquement.

11. Ecrire une fonction `change_rayon(entite, rayon)` et `change_masse(entite, masse)` qui affecte l'entité reçue en argument par effet de bords.

12. Ecrire une fonction `accelere(entite, angle, intensite)` qui ajoute à la vitesse de l'entité le vecteur $-\frac{intensite}{m} \times k_a \times \vec{u}$ où $\vec{u}$ est le vecteur unitaire formant un angle `angle` avec le vecteur $\vec{i}$ du repère. k_a est la constante d'accélération `ACCELERATION` définie de façon globale.

13. Ecrire une fonction `degaze(entites, entite, angle)` qui :
- retire une fraction de la masse de l'entité en utilisant la constante globale `DEGAZAGE` (la partie dégazée doit être supérieure à `EPSILON` sinon le dégazage ne peut pas avoir lieu)
- crée une nouvelle entité correspondant à la partie dégazée ayant la même vitesse et la même nature que l'entité et juxtaposée à l'entité au bout du rayon correspondant à l'angle en argument
- accélère les deux entités avec une intensité egale à la masse dégazée et avec les angles correspondants

14. Ecrire une fonction qui renvoie l'entité correspondant au joueur donné en argument dans la liste des entités.

15. Décommenter le corps de la fonction `click` et y incorporer l'appel à votre fonction précédente. Tester le jeu.

5 Interaction entre entités

16. Pour déterminer la taille de l'interface entre deux entités P_1 et P_2 de rayon R et r telles que $r < R$, on décide de convenir que la taille est $4r$ si P_2 est à l'intérieur de P_1, et $2h$ ou $4r - 2h$ suivant que P_2 est majoritairement à l'extérieur de P_1 ou majoritairement à l'intérieur (On se basera sur la position du centre de P_2 à l'intérieur ou à l'extérieur de P_1). h désigne la hauteur du triangle constitué des deux centres des entités et de l'un des points d'intersection des deux cercles (hauteur issue de ce point).

Pour calculer h, on utilise la formule de Héron pour calculer l'aire du triangle de côté a, b, c :

$$A = \sqrt{p(p-a)(p-b)(p-c)} \text{ avec } p = \frac{a+b+c}{2}$$

puis on en déduit h grâce à $A = \frac{b \times h}{2}$.

En déduire la fonction `taille_interface(entite1, entite2)`.

17. Ecrire une fonction `transfert(entites)` qui effectue tous les transferts de masse de la façon suivante :
- Trier les entités par rayon croissant : `entites.sort(key = rayon)`
- Pour chaque entité, considérer les entités plus petites qu'elles (c'est-à-dire celles d'indice plus petit)
- S'il y a collision, calculer le transfert théorique donné par le produit de la taille de l'interface et de la constante `TRANSFERT`
- Calculer le transfert effectif qui est égal au transfert théorique si celui-ci est bien compris entre 0 et la masse de l'entité la plus petite, sinon à 0 ou à la masse suivant les cas.
- Changer les masses en soustrayant à l'entité la plus petite le transfert effectif et en ajoutant ou soustrayant à l'entité la plus grosse le transfert effectif (selon qu'il s'agit d'un transfert entre entités de même nature ou de natures différentes).
- Supprimer toutes les particules de masse inférieure à `EPSILON`.

Dans la fonction `pas_de_jeu`, appeler `transfert(entites)`.

18. Ajouter dans la fonction `transfert` une accélération sur la particule qui absorbe l'autre. On appliquera une accélération dont l'angle est celui du vecteur vitesse de l'entité absorbée et dont l'intensité est égale à la masse transferée multipliée la norme du vecteur vitesse de l'entité absorbée.

19. Effectuer les modifications nécessaires pour utiliser des couleurs différentes suivant que les entités sont plus petites ou plus grandes que l'entité du joueur.