

CORRIGÉ : DÉTECTION DE COLLISIONS ENTRE PARTICULES (X PSI-PT 2016)

Rédigé par Jean-Pierre Becirspahic (jp.becir@info-llg.fr)

Partie I. Simulation du mouvement des particules

Question 1.

```
def deplacerParticule(particule, largeur, hauteur):
    x, y, vx, vy = particule
    if x + vx <= 0 or x + vx >= largeur:
        vx = -vx
    if y + vy <= 0 or y + vy >= hauteur:
        vy = -vy
    return (x + vx, y + vy, vx, vy)
```

Partie II. Représentation par une grille

Question 2.

```
def nouvelleGrille(largeur, hauteur):
    return [[None for j in range(hauteur)] for i in range(largeur)]
```

Question 3.

```
def majGrilleOuCollision(grille):
    largeur, hauteur = len(grille), len(grille[0])
    nvlleGrille = nouvelleGrille(largeur, hauteur)
    for i in range(largeur):
        for j in range(hauteur):
            if grille[i][j] is None:
                continue
            particule = grille[i][j]
            nvlleParticule = deplacerParticule(particule, largeur, hauteur)
            x, y, _, _ = nvlleParticule
            if nvlleGrille[int(x), int(y)] is not None:
                return None
            nvlleGrille[int(x), int(y)] = nvlleParticule
    return nvlleGrille
```

Question 4.

```
def attendreCollisionGrille(grille, tMax):
    t = 0
    while t < tMax and grille is not None:
        t += 1
        grille = majGrilleOuCollision(grille)
    if t == tMax:
        return None
    else:
        return t
```

Question 5. La fonction `deplacerParticule` est de complexité constante et la fonction `nouvelleGrille` de complexité en $O(\text{largeur} \times \text{hauteur})$. De ceci il résulte que la fonction `majGrilleOuCollision` est de complexité $O(\text{largeur} \times \text{hauteur})$: chaque case de la grille est scrutée à la recherche d'une particule, et quand une particule est trouvée on calcule en temps constant sa nouvelle position.

On en déduit que la complexité de la fonction `attendreCollisionGrille` est en $O(t\text{Max} \times \text{largeur} \times \text{hauteur})$.

Partie III. Représentation par liste de particules

Question 6.

```
def detecterCollisionEntreParticules(p1, p2):
    x1, y1, _, _ = p1
    x2, y2, _, _ = p2
    return (x2 - x1)**2 + (y2 - y1)**2 <= 2 * rayon
```

Question 7.

```
def maj(particules):
    largeur, hauteur, listeParticules = particules
    nvlleListe = []
    for p in listeParticules:
        nvlleListe.append(deplacerParticule(largeur, hauteur, p))
    return largeur, hauteur, nvlleListe
```

Question 8.

```
def majOuCollision(particules):
    nvlleParticules = maj(particules)
    _, _, liste = nvlleParticules
    for i in range(len(liste)-1):
        for j in range(i+1, len(liste)):
            if detecterCollisionEntreParticules(liste[i], liste[j]):
                return None
    return nvlleParticules
```

Question 9.

```
def attendreCollision(particules, tMax):
    t = 0
    while t < tMax and particules is not None:
        t += 1
        particules = majOuCollision(particules)
    if t == tMax:
        return None
    else:
        return t
```

La fonction `detecterCollisionEntreParticules` est de complexité constante et la fonction `maj` de complexité linéaire $O(n)$ où n est le nombre de particules puisque la fonction `deplacerParticule` est de complexité constante.

La détermination d'une collision entre deux particules possède une complexité en $O(n^2)$ puisqu'il y a $\frac{n(n-1)}{2}$ couples à examiner, donc la fonction `majOuCollision` a une complexité en $O(n^2)$.

Il en résulte que la complexité de la fonction `attendreCollision` est en $O(n^2 \times tMax)$.

Question 10. Entre les dates t et $t + 1$ une particule se déplace d'une distance inférieure ou égale à $vMax$ donc deux particules se rapprochent d'une distance inférieure ou égale à $2vMax$. Pour qu'elles se heurtent, il faut que leur distance à la date $t + 1$ soit inférieure ou égale à $2rayon$, et donc que leur distance à la date t soit inférieure ou égale à $2(rayon + vMax)$.

Question 11.

```
def majOuCollisionX(particules):
    nvlleParticules = maj(particules)
    _, _, liste = particules
    _, _, nvlleliste = nvlleParticules
    for i in range(len(liste)-1):
        for j in range(i+1, len(liste)):
            if liste[j][0] - liste[i][0] > 2 * (vMax + rayon):
                break
            if detecterCollisionEntreParticules(nvlleliste[i], nvlleliste[j]):
                return None
    return nvlleParticules
```

Dès lors que la différence entre les abscisses de deux particules excède $2(\text{rayon} + v\text{Max})$, il ne peut y avoir de collision à la date $t+1$.

Partie IV. Trier des listes partiellement triées

Question 12.

```
def scm(s):
    lst = []
    d, f = 0, 0
    while f < len(s) - 1:
        if s[f+1] >= s[f]:           # la scm se poursuit
            f += 1
        else:                         # la scm est terminée
            lst.append((d, f))
            d, f = f+1, f+1
    lst.append((d, f))
    return lst
```

Question 13. On commence par copier dans deux nouveaux tableaux les deux scm concernées, puis on fusionne la réunion de ces deux tableaux au sein du tableau s .

```
def fusionner(s, r1, r2):
    d1, f1 = r1
    d2, f2 = r2
    s1 = s[d1:f1+1]
    s2 = s[d2:f2+1]
    i, j = 0, 0
    for k in range(d1, f2+1):
        if j >= len(s2) or i < len(s1) and s1[i] <= s2[j]:
            s[k] = s1[i]
            i += 1
        else:
            s[k] = s2[j]
            j += 1
```

Question 14. On suppose bien entendu que les scm rangées dans la pile sont consécutives.

```
def depileFusionneRemplace(s, pile):
    while len(pile) > 1:
        d2, f2 = pile.pop()
        d1, f1 = pile.pop()
        fusionner(s, (d1, f1), (d2, f2))
        pile.append((d1, f2))
```

Question 15.

```
def alphaTri(s):
    listeScm = scm(s)
    pile = [listeScm[0]]
    for i in range(1, len(listeScm)): # première phase
        pile.append(listeScm[i])
        while len(pile) > 1:
            d2, f2 = pile.pop()
            d1, f1 = pile.pop()
            if (f1 - d1 + 1) < 2 * (f2 - d2 + 1):
                fusionner(s, (d1, f1), (d2, f2))
                pile.append((d1, f2))
            else:
                pile.append((d1, f1))
                pile.append((d2, f2))
                break
    depileFusionneRemplace(s, pile) # deuxième phase
```