DS d'informatique N°3 – Corrigé

Détection de collisions entre particules – corrigé

Partie I. Simulation du mouvement des particules

Question 1.

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

Partie II. Représentation par une grille

Question 2.

```
def nouvelleGrille(largeur, hauteur):
  G = []
  for i in range(hauteur)
    ligne = [None]*largeur
    G.append(ligne)
  return G
```

Question 3.

```
def majGrilleOuCollision(grille):
  largeur = len(grille[0])
  hauteur = len(grille)
  nG = nouvelleGrille(largeur, hauteur)   # Nouvelle grille vide
  for l in grille:
    for c in l:
      if c != None:
        nouvParticule = deplacerParticule(c, largeur, hauteur)
        nouvLigne = int(nouvParticule[0])
        nouvCol = int(nouvParticule[1])
        # Teste si la nouvelle case est vide
        if nG[nouvLigne][nouvCol] == None:
          nG[nouvLigne][nouvCol] = nouvParticule
        else:
          # Si elle n'est pas vide -> collision
          return None
  return nG
```

Question 4.

```
def attendreCollision(grille, tMax):
  for t in range(tMax):
    grille = majGrilleOuCollision(grille)
    if grille == None:
      return t+1
  return None
```

Question 5.

Dans le pire des cas (pas de collision), on effectue `tMax` itérations de la boucle `for`. La fonction `majGrilleOuCollision` parcourt toutes les cases de la grille, donc sa complexité est $O(largeur \times hauteur)$. La complexité de la fonction précédente est donc $O(largeur \times hauteur \times tMax)$.

Partie III. Représentation par une liste de particules

Listes non triées

Question 6.

```
def detecterCollisionEntreParticules(p1, p2):
  x1, y1, vx1, vy1 = p1
  x2, y2, vx2, vy2 = p2
  # carré de la distance entre les deux particules
  d2 = (x2-x1)**2 + (y2-y1)**2
  if d2 < (2*rayon)**2:
    return True
  else:
    return False
```

Question 7.

```
def maj(particules):
  largeur, hauteur, listePart = particules
  for i in range(len(listePart)):
    listePart[i] = deplacerParticule(listePart[i], largeur, hauteur)
  return (largeur, hauteur, listePart)
```

Question 8.

```
def majOuCollision(particules):
  particules  = maj(particules)
  liste = particules[2]
  for i1 in range(len(liste)-1):
    for i2 in range(i1+1, len(liste)):
      p1 = liste[i1]
      p2 = liste[i2]
      if detecterCollisionEntreParticules(p1, p2):
        return None
  return particules
```

Question 9.

```
def attendreCollision(particules, tMax):
  for i in range(tMax):
    particules = majOuCollision(particules)
    if particules == None:
      return i
  return None
```

La complexité de la fonction `majOuCollision` est $O(n^2)$ à cause des deux boucles imbriquées pour la détection de collisions. Donc la complexité de la fonction `attendreCollision` est $O(tMax \times n^2)$

Listes triées

Question 10.

Pour que deux particules `a` et `b` aient une chance de se rencontrer, elles ne doivent pas être éloignées de plus de $d = 2 \times \text{vMax} + 2 \times \text{rayon}$ car en une étape de simulation, elles se rapprochent au maximum de $2 \times \text{vMax}$ et pour qu'il n'y ait pas de collision elles doivent être écartées de $2 \times \text{rayon}$

Question 11.

```
def majOuCollisionX(particules):
  particules  = maj(particules)
  liste = particules[2]
  for i1 in range(len(liste)-1):
    i2 = i1+1
    while abs(liste[i2][0]-liste[i1][0])<2*(rayon+vMax):
      p1 = liste[i1]
      p2 = liste[i2]
      if detecterCollisionEntreParticules(p1, p2):
        return None
      i2 += 1
  return particules
```