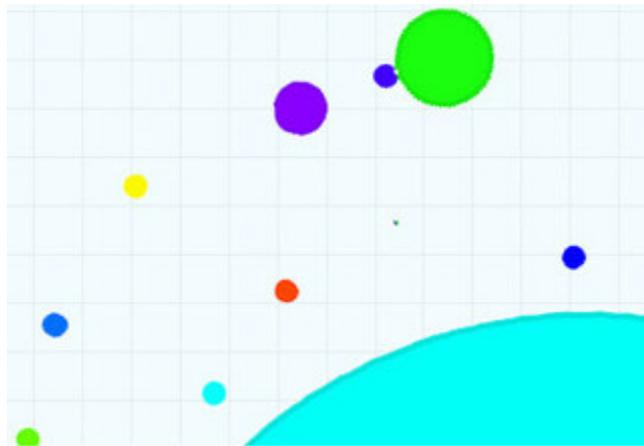


EXERCICE 2 (4 points)

Cet exercice porte sur les structures de données (programmation objet)

Dans un jeu de plateforme, des bulles de couleurs et de diamètres différents se déplacent de manière aléatoire. A chaque fois qu'une bulle touche une bulle plus grande, la petite cède son contenu à la plus grande, et donc celle-ci augmente de surface. Par exemple, si une bulle de 1 cm^2 rencontre une bulle de 4 cm^2 , la petite bulle disparaît et la plus grande a désormais une surface de 5 cm^2 . A chaque collision, la vitesse de la grande bulle est réduite de moitié.



Le développeur a choisi de coder en Python, chaque bulle est un objet disposant entre autre des attributs suivants :

- `xc`, `yc` sont deux entiers, les coordonnées du pixel placé au centre de la bulle,
- `rayon` est un entier, le rayon de la bulle en pixels,
- `couleur` est un entier, la couleur de la bulle,
- `dirx`, `diry` sont deux décimaux (float) qui déterminent les déplacements à l'horizontale et à la verticale à chaque fois que la bulle se déplace. Ces deux valeurs déterminent donc la direction et la vitesse de la bulle. Par exemple si `dirx` vaut `0.5` et `diry` vaut `0.0`, la bulle se déplace vers la droite uniquement alors que si `dirx` vaut `-1.0` et `diry` vaut `0.0`, la bulle se déplace vers la gauche et deux fois plus vite que précédemment.

On suppose que toutes les fonctions de la bibliothèque `math` ont déjà été importées par l'instruction `from math import *`.

La fonction `randint` de la bibliothèque `random` prend en paramètre deux entiers et renvoie un entier aléatoire dans la plage définie par les deux paramètres.

Exemple : `randint(-1, 5)` peut renvoyer une des valeurs suivantes : `-1, 0, 1, 2, 3, 4, 5`.

1. Pour simplifier, on se limitera à un jeu de six bulles. Au départ, on crée une liste appelée `Mousse` de longueur six contenant six emplacements vides :

```
Mousse = [None, None, None, None, None, None]
```

Le code ci-dessous montre le début du programme et notamment la structure définition de la classe nommée `Cbulle` ainsi que le code permettant le déplacement d'une bulle.

```
from random import randint
from math import *

class Cbulle:
    def __init__(self):
        self.xc = randint(0, 100)
        self.yc = randint(0, 100)
        self.rayon = randint(0, 10)
        self.dirx = float(randint(-1, 1)) # dirx et diry valent
        self.diry = float(randint(-1, 1)) # -1.0 ou 0.0. ou 1.0
        self.couleur = randint(1, 65535)

    def bouge(self):
        # déplace la bulle
        self.xc = self.xc + self.dirx
        self.yc = self.yc + self.diry
```

On crée les six bulles une à une et ces objets sont stockés dans les emplacements vides de la liste `Mousse`.

```
Mousse = [bulle1, bulle2, bulle3, bulle4, bulle5, bulle6]
```

Lors d'une collision, la bulle la plus petite disparaît et est remplacée dans la liste par la valeur `None` tandis que la plus grosse a sa surface qui augmente.

Au cours d'une partie, si une ou plusieurs bulles ont disparu, le programme peut en introduire de nouvelles dans le jeu: dans ce cas, lorsqu'une nouvelle bulle apparaît, elle remplace le premier `None` de la liste `Mousse`.

- a. Recopier les quatre dernières lignes et compléter les `.....` du code python ci-dessous.

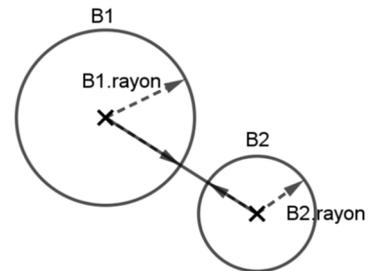
```
def donnePremierIndiceLibre(Mousse):
    """
    Mousse est une liste.
    La fonction doit renvoyer l'indice du premier
    emplacement libre (contenant None) dans la liste Mousse
    ou renvoyer 6 en l'absence d'un emplacement libre dans
    Mousse.
    """
    i = 0
    while ..... and Mousse[i] != None :
        .....
    return i
```

b. Lorsque le jeu crée une bulle (instance de la classe `Cbulle`), il doit ensuite la placer dans la liste `Mousse` à la place d'un `None`.

Ecrire la fonction `placeBulle(B)` qui reçoit en paramètre un objet de type `Cbulle` et qui place cet objet dans la liste `Mousse`. Cette fonction ne renvoie rien, mais la liste `Mousse` est modifiée. Si aucun emplacement n'est disponible, la fonction ne modifie rien.

2. Pour le bon déroulement du jeu, on a besoin aussi d'une fonction `bullesEnContact(B1, B2)` qui renvoie `True` si la bulle `B2` touche la bulle `B1` et `False` dans le cas contraire.

On peut remarquer que deux bulles sont en contact si la distance qui sépare leur centre est inférieure ou égale à la somme de leurs rayons.



On dispose de la fonction `distanceEntreBulles(B1, B2)` qui calcule et renvoie la distance entre les centres de bulles `B1` et `B2`.

Ecrire la fonction `bullesEnContact(B1, B2)`.

3. Quand une petite bulle touche une plus grosse bulle, on appelle la fonction `collision`, ci-dessous, où `indPetite` est l'indice de la petite bulle et `indGrosse` l'indice de la grosse bulle dans `Mousse`.

Recopier et compléter les `.....` de la fonction `collision`.

```
def collision(indPetite, indGrosse, mousse) :
    """
    Absorption de la plus petite bulle d'indice indPetite
    par la plus grosse bulle d'indice indGrosse. Aucun test
    n'est réalisé sur les positions.
    """
    # calcul du nouveau rayon de la grosse bulle
    surfPetite = pi*Mousse [indPetite].rayon**2
    surfGrosse = pi*Mousse [indGrosse].rayon**2
    surfGrosseApresCollision = .....
    rayonGrosseApresCollision = sqrt(surfGrosseApresCollision/pi)
    #réduction de 50% de la vitesse de la grosse bulle
    Mousse[indGrosse].dirx = .....
    Mousse [indGrosse].diry = .....
    #suppression de la petite bulle dans Mousse
    .....
```