Rapport de TP de programmation concurrente

Grégory Hubert, L3 informatique

10 novembre 2018

Résumé

Dans ce rapport je vais vous présenter un code en python qui permet de réaliser l'exercice 3.2 du TP de programmation concurrente. En effet nous allons observer le code permettant de faire rebondire des boules dans un interface graphique.

1 Les boules

Nous allons parler içi de la création des boules de ce projet. Ces boules devront ricocher contre les bords de la fenêtre et de plus disparaitre en cas de collision entre deux boules. De plus j'ai rajouter le fait que ces boules auront une taille aléatoire et une couleur aléatoire lors de leur création.

1.1 Initialisation de la classe

```
ronds = list()

    def __init__(self,rond,canvas,fenetre,x,y,color = choice(colors)):
        boule.ronds.append(self)
        self.fenetre=fenetre
        self.canvas=canvas
        self.rond = rond
        self.rayon=randint(10,20)
        self.x=x
        self.y=y
        self.dx=randint(-5,5)
        self.dy=randint(-5,5)
        while self.dx == 0 and self.dy == 0:
            self.dx=randint(-5,5)
            self.dy=randint(-5,5)
        self.move()
```

On crée ainsi une liste dans laquel seront ranger les boules afin de faire les test de collision sur eux plus tard. On donne en paramètre la fenêtre et canvas afin d'afficher les boules dans la fenêtre et leur position en x et y qui auront été choisi de manière aléatoire comme la couleur. On lance ensuite le mouvement de la boule ainsi crée avec la fonction move().

1.2 Les collisions

```
def collision(self, other, fenetre):
        if (self.x+self.rayon>other.x-other.rayon and self.x-self.rayon<other.x+other.rayon and self
            fenetre.score += 1
            fenetre.scoring['text'] = ('Score: {}'.format(fenetre.score))
            self.canvas.delete(self.fenetre,self.rond)
```

```
            boule.ronds.remove(self)
            self.canvas.delete(self.fenetre,other.rond)
            boule.ronds.remove(other)
```

Cette fonction nous permet alors de tester la collision qu'il peut y avoir entre deux boules en testant si leur hitbox se touchent l'une à l'autre. Si tel est le cas, nous incrémentons le score de 1 et faisons disparaitre les deux boules du tableau et de la fenêtre qui les contiennent.

1.3 Le mouvement des boules

```
def move(self):
        self.canvas.move(self.rond,self.dx,self.dy)
        self.x += self.dx
        self.y += self.dy
        if(self.x>h or self.x<0):
            self.dx=-self.dx
        if(self.y>l or self.y<0):
            self.dy=-self.dy
        self.fenetre.after(10,self.move)
```

Cette fonction indique le mouvement que devra prendre la boule qui aura été choisi de manière aléatoire à la fois en x et en y. De plus si la boule dépasse la taille de la fenêtre elle dois alors changer de sens en x ou y en fonction de qu'elle bord elle aura touché. Cette fonction s'appelle elle même tous les 10ms afin que le mouvement continue jusqu'a une collision ou disparition de la boule.

1.4 L'arrêt des boules

```
def stop(self):
        self.dx = 0
        self.dy = 0
```

Lorsque le jeu sera mis en pause, les boules seront stoper par cette fonction qui leur idiquera de rester sur place.

2 La fonction main

```
def __init__(self):
        Thread.__init__(self)
        self.fenetre = Tk()
        self.score = 0
        self.pause = 1
        self.t = 0
        self.scoring = Label(self.fenetre, text = 'Score {}'.format(self.score))
        self.scoring.pack()
        self.t_temps = Label(self.fenetre, text = 'Temps {}'.format(self.t))
        self.t_temps.pack()
        self.canvas = Canvas(self.fenetre, width = h, height = l, bg = 'black')
        self.canvas.pack()
        self.ajout = Button(self.fenetre, text = '+', command = self.ajout)
        self.ajout.pack(side = LEFT)
        self.effacer = Button(self.fenetre, text = '-', command = self.effacer)
        self.effacer.pack(side = RIGHT)
        self.p = Button(self.fenetre, text = 'Pause', command = self.pau)
```

```
        self.p.pack(side = TOP)
        self.s = Button(self.fenetre, text = 'Start', command = self.star)

    def run(self):
        pass
```

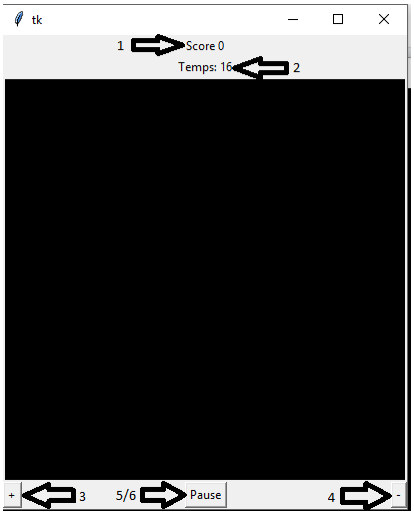
On initialise tous ce qu'il y a pour faire fonctionner le jeu :

— la fenêtre

— les boutons score, temps, ajout, retrait de boule, pause et start ainsi que leur position dans la fenêtre

Voici la fenêtre qu'on obtient avec les différents bouton :

1. Le score
2. Le temps
3. le bouton ajout de boule
4. le bouton retrait de boule
5. le bouton pause (içi afficher)
6. le bouton start (içi caché en attente du jeu en pause)



Nous allons voir maintenant les différentes fonction associé aux boutons.

2.1 Ajout de boule

```
def ajout(self):
        rayon = randint(10,30)
        x = randint(rayon, h-rayon)
        y = randint(rayon, l-rayon)
        color = choice(colors)
        rond = self.canvas.create_oval(x,y,x+rayon,y+rayon, fill = color)
        boule(rond,self.canvas,self.fenetre,x,y,color)
        self.pause = 1
        self.star()
```

Dans cette fonction nous donnons les valeur du rayon et de la position de la boule aléatoire ainsi que sa couleur puis nous la créons. Cette fonction est appelé par le bouton "+" du jeu et en cas de pause relance le jeu.

2.2 Retrait de boule

```
def effacer(self):
        if boule.ronds != []:
            lcanvas = self.canvas.find_all()
            self.canvas.delete(lcanvas[len(lcanvas)-1])
            boule.ronds.pop(len(boule.ronds)-1)
```

Dans cette fonction, nous faisons disparaitre la dernière boule à avoir été ajouté du tableau qui le contient ainsi que dans la fenêtre. Cette fonction est appelé par le bouton "-" du jeu.

2.3 Pause du jeu

```
def pau(self):
            if self.pause:
                for i in boule.ronds:
                    boule.stop(i)
                    self.pause = 0
                    self.p.pack_forget()
                    self.s.pack(side = TOP)
```

Dans cette fonction, nous faisons en sorte que les boules ne bouge plus via la fonction stop 1.4. De plus on affichera alors le bouton "Start" au moment ou le bouton "Stop" qui appelle cette fonction est appuyé tandis que ce même bouton sera lui caché.

3 Le temps

```
class chrono():
    def __init__(self,t,fenetre):
        self.t = t
        self.fenetre = fenetre
        self.affiche()

    def affiche(self):
        if f.pause:
            f.t_temps['text'] = ('Temps: {}'.format(self.t))
            self.t += 1
            self.fenetre.after(1000, self.affiche)
        if f.pause == 0:
            self.t = self.t
            self.fenetre.after(1000, self.affiche)
```

Cette classe permet de crée un chronomètre qui affiche le nombre de seconde passé depuis le lancement du jeu et qui s'arrêtera lorsque le jeu sera mise en pause et repartira quand le jeu repartira aussi.

4 Le lancement du jeu

```
class calcul(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.pause = True
        self.t = 0
```

```
    def run(self):
        while 1:
            if self.pause:
                for i in boule.ronds:
                    for j in boule.ronds:
                        if j != i:
                            i.collision(j,f)
            time.sleep(0.01)

f = main()
f.start()
c = chrono(f.t,f.fenetre)
calcul_f = calcul()
calcul_f.start()
f.fenetre.mainloop()
```

Dans la classe calcul, nous lançons le jeu , c'est à dire tester la collision de chaque boule. Après avoir crée toutes ces fonctions et classe, nous lançons alors le jeu via ces quelques dernière lignes.