Modélisations autour de la Formule 1

Corrigé UPSTI

I Modélisation sommaire d'un circuit

I.A – Validité de la représentation d'un circuit

Question 1

```
def longueur1(c, d):
    return c.count("A")*d
```

Question 2

Les étapes liées à la boucle for sont les suivantes, et le resultat final est ["A","A","D","A"].

Etape	e	res	nbg
1	"A"	["A"]	0
2	"A"	["A","A"]	0
3	"G"	["A","A"]	1
4	"D"	["A","A"]	0
5	"G"	["A","A"]	1
6	"G"	["A","A"]	2
7	"G"	["A","A"]	3
8	"A"	["A","A","D","A"]	0

Question 3 Le but de ce code est de simplifier les successions de virage non significatives. En effet pour passer d'une orientation à une autre il ne peux pas y avoir plus de 2 instructions de virage (90 ° à gauche, 180 ° à gauche, 270 ° à gauche (ce qui revient en fait à faire 90 ° à droite)), On additionne donc les virages demandés pour simplifier !

Question 4

```
def contient_demi_tour1(c):
    cmin = representation_minimale(c)
    for i in range(len(cmin)-1) :
        # Test de la présence de demi-tour
        if cmin[i:i+2] == ["G","G"] :
            return True
    return False
```

Question 5

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

```python
def est_ferme1(c):
    # definition des coordonnees initiales
    x,y,ori = 0,0,0
    # Variation de position en fonction de l'orientation
    dx = [1,0,-1,0]
    dy = [0,1,0,-1]
    for elt in c :
        if elt == "A" : # On avance
            x+=dx[ori]
            y+=dy[ori]
        elif elt == "G" : # On change l'orientation
            ori = (ori + 1)%4
        else : # Virage à droite
            ori = (ori - 1)%4
    return (x,y,ori) == (0,0,0) # On regarde si on est dans la configuration d'origine
```

Question 6

```python
def circuit_convenable1(c):
    # Sur la même base que la question précédente
    if contient_demi_tour1(c) or not est_ferme1(c) :
        return False
    # definition des coordonnees initiales
    x,y,ori = 0,0,0
    # Variation de position en fonction de l'orientation
    position=[]
    dx = [1,0,-1,0]
    dy = [0,1,0,-1]
    for elt in c :
        if elt == "A" : # On avance
            x+=dx[ori]
            y+=dy[ori]
            if [x,y] in position : # On est deja passé par ce point
                return False
            position.append([x,y])
        elif elt == "G" : # On change l'orientation
            ori = (ori + 1)%4
        else : # Virage à droite
            ori = (ori - 1)%4
    return True
```

I.B – Tracé d'un circuit

Question 7

```python
def dessine_circuit1():
    for elt in c :
        if elt == "A" :
            turtle.forward(d)
        elif elt == "G" :
            turtle.left(90)
        else :
            turtle.right(90)
```


© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

II Modélisation plus réaliste d'un circuit

II.A – Validation

Question 8

```
def element_valide2(e):
    # L'element est un entier
    if isinstance(elt,int) :
        return  elt > 0 :
    # L'element est une liste (virage)
    elif isinstance(elt,tuple) :
        r, a = elt
        if isinstance(r,int) and isinstance(a,int) :
            return (r > 0) and (-360 < a < 360)
    return False
```

II.B – Première méthode de tracé à l'écran

Question 9

```
def dessine_circuit2(c, echelle):
    for elt in c :
        if isinstance(elt,int) :  # Tout droit
            turtle.forward(echelle*elt)
        elif isinstance(elt,tuple) :
            if elt[1] < 0 : # A droite
                r = -elt[0]
            else :
                r = elt[0]
            turtle.circle(r*echelle,abs(elt[1]))
```

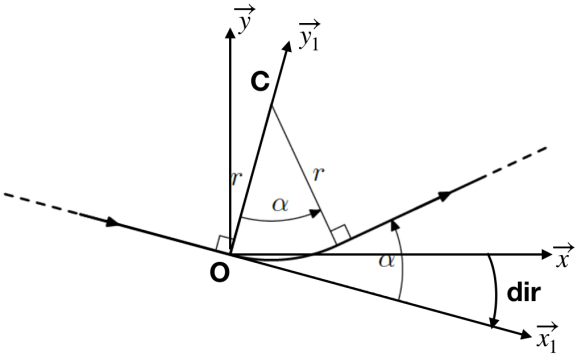
II.C – Tracé pixel par pixel

Question 10

```
def matrot(t) :
    tr = t*np.pi/180 # Conversion en radians
    return np.array([[np.cos(tr),-np.sin(tr)],[np.sin(tr),np.cos(tr)]])
```

Question 11

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI



Les entrées sont :

- `pos` est un array numpy représentant la position du point d'origine de la courbe (O) ;
- `dir` est un nombre représentant un angle (en degrés) représentation l'inclinaison d'incidence ;
- `r` est un nombre représentant le rayon de courbure ;
- `alpha` est un nombre signé représentant l'angle de la portion de courbe.

La sortie renvoie alors la position du centre de courbure (C) de la courbe de rayon `r` et d'angle `alpha`.

$\overrightarrow{OC} = r \cdot \overrightarrow{y_1}$. Pour trouver les coordonnées du point C, il faut donc ajouter aux coordonnées du point O, le vecteur $r \cdot \overrightarrow{y_1}$ écrit dans la base $(\overrightarrow{x}, \overrightarrow{y})$. On applique donc la matrice rotation (d'angle dir) au vecteur $\overrightarrow{y_1}$. La position du centre dépend du sens de rotation d'où la présence du signe.

Question 12

```
def dessine_circuit3(s, c, echelle):
    position = np.array([len(s)//2,len(s[0])//2]) # Position du centre de l'écran
    dir = 0
    for elt in c :

        if isinstance(elt,int) :  # Tout droit
            delta = matrot(dir)@np.array([elt*echelle,0.]) # Projection dans la base x,y
            ligne(s,position,position+delta*echelle)
            position+=delta*echelle # Mise à jour de l'extremité
        elif isinstance(elt,tuple) :
            centre = cc(position,dir,echelle*elt[0],elt[1]) # calcul du centre
            arc(s,position,centre,elt[1]) #dessin de l'arc
            dir+=elt[1] # Ajout de l'angle
            vecteur = centre - position
            position = centre - matrot(elt[1]) @ vecteur # Rotation du segment point d'origine -> centre
```

III Le parcours d'une voiture

III.A – Formules de calcul des vitesses et temps de parcours

III.A.1)

Question 13 L'expression temporelle de la vitesse lors d'une accélération constante est de la forme $v(t) = a_{max} \cdot t$. Le temps nécessaire est alors $t_{vmax} = \dfrac{v_{max}}{a_{max}} = \dfrac{100}{10} = 10\,\text{s}$.

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

Question 14 Pour une accélération on a $v_2 = a_{max} \cdot t_{min} + v_1$ donc $t_{min} = \dfrac{v_2 - v_1}{a_{max}}$.

Pour une décélération on a alors $v_2 = f_{max} \cdot t_{min} + v_1$ donc $t_{min} = \dfrac{v_2 - v_1}{f_{max}}$.

Application numérique : $t_{min} = \dfrac{120 - 300}{3.6 \times (-20)} = 2,5\,\text{s}$

Question 15 En intégrant la relation de la question précédente : $d = a_{max} \cdot \dfrac{{t_{min}}^2}{2} + v_1 \cdot t_{min}$.

On remplace alors l'expression de t_{min} d'où $d = a_{max} \cdot \dfrac{(v_2 - v_1)^2}{2\,{a_{max}}^2} + v_1 \cdot \dfrac{v_2 - v_1}{a_{max}}$

$$= \frac{{v_2}^2 - 2\,v_2\,v_1 + {v_1}^2 + 2\,v_1\,v_2 - 2\,{v_1}^2}{2\,a_{max}} = \frac{{v_2}^2 - {v_1}^2}{2\,a_{max}}.$$

Question 16

De la même manière : $d = \dfrac{{v_2}^2 - {v_1}^2}{2\,f_{max}} = \dfrac{120^2 - 300^2}{3,6^2 \times 2 \times (-20)} \simeq 146\,\text{m}$

III.A.2)

Question 17 Il faut le temps d'accélérer jusqu'à v_{max} puis de décélérer pour atteindre v_2 :

$$d_{min} = \frac{{v_{max}}^2 - {v_1}^2}{2\,a_{max}} + \frac{{v_2}^2 - {v_{max}}^2}{2\,f_{max}}$$

Question 18 On est sur un profil en trapèze :

- accélération : en $t_a = \dfrac{v_{max} - v_1}{a_{max}}$ on parcourt $\dfrac{{v_{max}}^2 - {v_1}^2}{2\,a_{max}}$;
- décélération : en $t_d = \dfrac{v_2 - v_{max}}{f_{max}}$ on parcourt $\dfrac{{v_2}^2 - {v_{max}}^2}{2\,f_{max}}$;
- à vitesse constante v_{max} il reste à parcourir : $d - d_{min}$ en $t_c = \dfrac{d - d_{min}}{v_{max}}$.

Le temps nécessaire est alors $t = t_a + t_c + t_d$

Question 19

- Accélération : $t_a = \dfrac{v_3 - v_1}{a_{max}}$;
- décélération : $t_d = \dfrac{v_2 - v_3}{f_{max}}$.

Le temps nécessaire est alors $t = t_a + t_d$

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

III.B – Implantation en Python

III.B.1) Temps pour un tour

Question 20

```python
def vitesses_entree_max(c,vf) :
# Il faut parcourir la liste à l'inverse pour tenir compte de la remarque sur la vitesse ultérieure
# En entrée de ligne droite il n'y a pas de limitation de vitesse
    v= [vf]
    for elt in c[::-1] :
        if isinstance(elt,int) :
        # En ligne droite on utilise la formule de la distance de freinage pour trouver la vitesse max
            v.append(min(VMAX,2*FMAX*elt+v[-1]**2))
        else :
        # En virage, vitesse minimale entre la vitesse en sortie de virage et celle autorisée par le virage
            v.append(min(v[-1],vr(elt[0])))
    return v[:0:-1] # On retourne la liste
```

Question 21

```python
def temps_droite (d,v1,v2) :
    # On utilise les notations de la Q19
    v3 = vmax_droite(d,v1,v2)
    ta = (v3-v1)/AMAX
    da = (v3**2-v1**2)/(2*AMAX)
    if v3 < v2 : # Acceleration pure
        return ta, v3
    td = (v2-v3)/FMAX
    dd = (v2**2-v3**2)/(2*FMAX)
    if dd > d : # Deceleration trop longue
        raise ValueError
    return ta+td+(d-da-dd)/VMAX,v2
```

Question 22

```python
def temps_tour (c,v0,vf) :
    # On calcule les vitesses d'entrée :
    lv = vitesses_entree_max(c,vf)
    lv[0] = v0 # la vitesse de depart est égale à v0
    lv.append(vf)# Pour le calcul de dernier segment
    t = 0
    for i,elt in enumerate(c) :
        if isinstance(elt,int) :
        #Si on est en ligne droite
            td,vd = temps_droite(elt, lv[i], lv[i+1])
            t+=td
            lv[i+1] = vd # Mise à jour de la vitesse en bout de droite
        else :
            # distance = rayon * angle (en radians)
            t+= elt[0]*np.radians(elt[1])/lv[i]
    return t
```


© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

III.B.2) Temps de course

Question 23

```
def temps_course(c,n) :
    # Concatenation de n tours en un circuit unique, puis appel de temps_tour

    return temps_tour(n*c,0,VMAX)
```

Question 24 La longueur de la liste est $n \times c$. La fonction `temps_tour` est de complexité linéaire (un appel à la fonction `vitesses_entree_max` qui est linéaire et une simple boucle for). La complexité est donc $\mathcal{O}(n \times c)$.

IV Gestion des résultats

Question 25 Il peut y avoir 2 pilotes qui ont le même nom de famille !

Question 26

70 ans de Grand Prix, avec 20 grands Prix par an, 20 pilotes par course et 60 tours (en moyenne) par courses.

Cela donne : $70 \times 20 \times 20 \times 60 \approx 1,7 \cdot 10^6$ tours

Question 27

SELECT gp_date,ci_nom
FROM GranPrix **JOIN** Circuit **ON** GrandPrix.ci_id = Circuit.ci_id
WHERE ci_pays = 'France'
ORDER BY gp_date

Question 28

SELECT ci_nom, pi_nom, sum(to_temps)
FROM Participation **JOIN** Tour **ON** Participation.pa_id = Tour.pa_id
JOIN GrandPrix **ON** GrandPrix.gp_id = Cla.gp_id
JOIN Pilole **ON** Pilote.pi_id = Cla.pi_id
JOIN Circuit **ON** Circuit_id = GrandPrix.ci_id
GROUP BY gp_id,pi_id
HAVING pa_cla = 1 AND EXTRACT('year' FOR gp_date) = 2021

Question 29

La requête renvoie le nom du circuit, le nom du pilote, la date du grand prix et le temps du meilleur tour.

Le résultat de cette requête est donc le record historique de chacun des circuits ainsi que le pilote qui a réalisé cette performance.

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI