freinage.py

```python
'''Programme qui permet d'obtenir les paramètres d'un modélisation polynomiale à partir d'un jeu de valeurs'''

#Importation des modules
import numpy as np
import matplotlib.pyplot as plt

#Conversion dictionnaire -> listes
def conversion(dictionnaire : dict) -> tuple:
    vitesses = list(dictionnaire.keys())
    vitesses.sort()
    distances = [dictionnaire[vitesse] for vitesse in vitesses]
    return vitesses,distances

#Détermination du polynôme
def calculPolynome(vitesses : list, distances : list, degre : int) -> np.ndarray:
    return np.polyfit(vitesses, distances, degre)

def calculPolynomeListe(vitesses : list, distances : list, degre : int) -> list:
    polynome = calculPolynome(vitesses, distances, degre)
    polynome = [e for e in polynome]
    polynome.reverse()
    return polynome

def calculPolynomeDict(dictionnaire : dict, degre : int) -> list:
    vitesses, distances = conversion(dictionnaire)
    return calculPolynomeListe(vitesses, distances, degre)

#Calcul de la courbe des valeurs
def distance(polynome : list, vitesse : float) -> float:
    dist = 0
    taille = len(polynome)
    for j in range(taille):
        dist += polynome[j] * vitesse**(taille-1-j)
    return dist

#Fonction d'affichage
def affichage(vitesses : list, distances : list, degre : int, incertitudeVitesse : float, incertitudeDistance : float, label : str) -> None:
    polynome = calculPolynome(vitesses, distances, degre)
    vitessesAff = [i for i in range(40)]
    distancesAff = [distance(polynome, vitesse) for vitesse in vitessesAff]
    print(distancesAff)
    plt.figure(figsize=(8, 6), dpi=150)
    for i in range(len(vitesses)):
        plt.errorbar(vitesses[i], distances[i], xerr = incertitudeVitesse*vitesses[i], yerr = incertitudeDistance, fmt = "+", color='blue')
    plt.plot(vitessesAff, distancesAff, color = 'green', label = label) #Affichage de la courbe du polynôme
    plt.ylabel("Distance de freinage (m)")
    plt.xlabel("Vitesse (km/h)")
    plt.legend()
```

```
    plt.show()

def affichageDict(dictionnaire : dict, degre :  int, incertitudeVitesse : float, incertitudeDistance : float, label : str) -> None:
    vitesses, distances = conversion(dictionnaire)
    affichage(vitesses, distances, degre, incertitudeVitesse, incertitudeDistance, label)

def comparaison(v1 : list, d1 : list, l1 : str, v2 : list, d2 : list, l2 : str, v3 : list, d3 : list, l3 : str, v4 : list, d4 : list, l4 : str) -> None:
    plt.figure(figsize=(8, 6), dpi=150)
    plt.plot(v1, d1, label = l1)
    plt.plot(v2, d2, label = l2)
    plt.plot(v3, d3, label = l3)
    plt.plot(v4, d4, label = l4)
    plt.ylabel("Distance de freinage (m)")
    plt.xlabel("Vitesse (km/h)")
    plt.legend()
    plt.show()
```

constantes.py

```
import freinage

#Echelle de longueur
pixel = 50 #Nombre de pixels pour 1 mètre
case = 11 #Nombre de pixels pour une case

#Echelle d'un vélo
nombreCase = [2,8] #2 cases de largeur et 8 cases de longueur

#Echelle de temps
dtms = 300 #Intervalle d'actualisation en ms
dt = dtms/10**3 #Intervalle d'actualisation en s

#Variables concernant la course
largeurRoute = 6

#Variables concernant la fenêtre en nombres de cases
largeur = 150
hauteur = largeurRoute * pixel // case

#Incertitudes
incertitudeVitesse = 0.02
incertitudeDistance = 0.5

#Importation des vitesses et des distances de freinage mesurés
valeursDisquesS = {0: 0, 22.9: 4.45, 18.4: 3.20, 25.1: 5.25, 30.9: 7.53, 5.1 : 0.56, 10.4: 1.03, 24.2: 4.57, 13: 2.18, 31.7: 7.65, 36: 9.46, 26.1: 5.95, 26.4: 6.42, 18.4: 3.42, 35.9: 9.99, 38.4: 11.86, 21.2: 4.65, 24.8: 5.46, 30.8: 8.13}
valeursDisquesM = {0: 0, 19: 3.99, 23.5: 5.30, 25.5: 4.89, 23.3: 5.34, 27.2: 5.28, 23.1 : 5.07, 24.8 : 4.81, 5.1 : 0.56, 12.3 : 2.1, 38.5: 12, 31.8: 7.72, 34.4: 9.88, 31 : 7.65, 28.9 :6.91}
valeursPatinsS = {0: 0, 32: 8.12, 35: 11.27, 37.3: 11.5, 5.5: 1.14, 20.6: 3.61, 22.9: 5.2, 7.5: 1.2, 33.3: 9.08, 32.6: 8.75, 36.1: 10.42, 26.4: 6.46, 35.6: 9.8, 37.6: 13.6, 19.3: 3.98, 28.7: 6.79, 16.8: 2.78, 25.5: 5.83, 12.6: 2.09}
valeursPatinsM = {0: 0, 24.3: 6.75, 30.3: 9.56, 29: 9.30, 24.8: 6.42, 33: 11.51, 18.9: 4.99, 36.3: 12.96, 38: 13.12, 24.1: 5.34, 35.9 : 12.81, 17.1 : 3.98, 33.2 : 10.51, 5.2 : 1.2, 19.5 : 5.01}

#Polynômes freinage
disquesS = freinage.calculPolynomeDict(valeursDisquesS, 6)
disquesM = freinage.calculPolynomeDict(valeursDisquesM, 3)
patinsS = freinage.calculPolynomeDict(valeursPatinsS, 5)
patinsM = freinage.calculPolynomeDict(valeursPatinsM, 3)
temps = 'Sec'
```

main.py

```
#Importation des modules
from tkinter import*
from tkinter import ttk
from tkinter.messagebox import*
from math import sqrt,ceil,floor
from random import randint
import constantes as cst

def vider() -> None:
    fichier = open('flotte.txt','w')
    fichier.seek(0)
    fichier.write('')
    fichier.close()

#Obtention des résultats
resultatLargeur = False
resultatObstacle = False
resultatFrein = False

if resultatLargeur:
    fichierLargeur = open("hauteur.txt","r")
    hauteur = int(fichierLargeur.read())
    fichierLargeur.close()
    if hauteur == 22:
        vider()
else:
    hauteur = cst.hauteur

if resultatObstacle:
    fichierObstacle = open("obstacle.txt","r")
    obstacle = fichierObstacle.read() == "True"
    fichierObstacle.close()
    if not obstacle:
        vider()

if resultatFrein:
    fichierProportion = open("proportion.txt",'r')
    proportion = int(fichierProportion.read())
    fichierProportion.close()
    if proportion == 0:
        vider()

#Initialisation de la fenêtre
fen = Tk()
fen.title('TIPE')
can = Canvas(fen,width = cst.largeur * cst.case,height = hauteur * cst.case)
can.pack()

#Variables principales
listeVelo = [] #Liste qui contiendra tous les vélos
fenVelo = None #Variable qui contiendra la fenêtre des paramètres de création
```

```
52
53  def convertirVitesse(vitesse : float) -> float:
54      '''Fonction qui convertir la vitesse en argument en km/h en m/s'''
55      return vitesse * 10**3/3600
56
57  #Récupération de la liste des vélos déjà crées
58  fichierVelo = open('flotte.txt','r')
59  lignesVelo = fichierVelo.readlines()
60  fichierVelo.close()
61
62  #Classe cycliste
63  class cycliste:
64      def __init__(self : object, vitesse : int, frein : str, case : list, chute = False) ->
    None :
65          self.vitesse = vitesse
66          self.frein = frein
67          self.couleur = 'black'
68          self.bordure = 'red'
69          self.case = case
70          self.chute = False
71          #Variables liées au freinage
72          self.freinageEnCours = False
73          self.vitesseInitiale = self.vitesse
74          self.iterations = None
75          self.iterationsRestantes = None
76          self.creation()
77          if chute:
78              self.chuter()
79      def creation(self : object) -> None:
80          #On crée un vélo sur l'ensemble de carrés avec le coin gauche aux cases dans la liste
    'case'
81          # et le coin droit décalé du 'nombreCase'
82          x1, y1, _, _ = can.coords(grille[self.case[0]][self.case[1]])
83          _, _, x2, y2 = can.coords(grille[self.case[0]+cst.nombreCase[0]-1]
    [self.case[1]+cst.nombreCase[1]-1])
84          self.id = can.create_rectangle(x1,y1,x2,y2, fill = self.couleur, outline =
    self.bordure,width=3, tag = 'velo')
85      def chuter(self : object) -> None:
86          self.chute = True
87          self.couleur = 'red'
88          can.itemconfig(self.id, fill = self.couleur)
89      def supprimer(self : object) -> None:
90          '''Fonction qui supprime le vélo de la liste'''
91          global listeVelo, fenVelo
92          can.delete(self.id)
93          listeVelo.remove(self)
94          fenVelo.destroy()
95      #Fonctions qui permettent de déplacer un vélo avec les touches du clavier
96      def decaleGauche(self : object) -> None:
97          if self.case[1] > 0 and creationPossible(str(self.case[0]),str(self.case[1]-1),
    [self]):
98              self.case[1] -= 1
99              can.move(self.id,-cst.case,0)
100     def decaleDroite(self : object) -> None:
```

```
        if self.case[1] < cst.largeur and
creationPossible(str(self.case[0]),str(self.case[1]+1),[self]):
            self.case[1] += 1
            can.move(self.id,cst.case,0)
    def decaleHaut(self : object) -> None:
        if self.case[0] > 0 and creationPossible(str(self.case[0]-1),str(self.case[1]),
[self]):
            self.case[0] -= 1
            can.move(self.id,0,-cst.case,)
    def decaleBas(self : object) -> None:
        if self.case[0] < hauteur and
creationPossible(str(self.case[0]+1),str(self.case[1]),[self]):
            self.case[0] += 1
            can.move(self.id,0,cst.case,)
    def freinage(self : object) -> None:
        if not self.freinageEnCours: #Si le vélo n'est pas en train de freiner
            self.freinageEnCours = True
            self.vitesseInitiale = self.vitesse
            distance = self.calculDistanceFreinage() #On calcule la distance nécessaire pour
s'arrêter
            #On détermine le nombre d'itérations nécessaires pour s'arrêter complétement
            iterFloat = 2*distance/(cst.dt*convertirVitesse(self.vitesseInitiale))
            self.iterations = round(iterFloat) + 1
            self.iterationsRestantes = self.iterations - 1
        #A chaque itération, on diminue la vitesse en fonction
        # du nombre d'itérations restantes et de la vitesse initiale
        if self.iterationsRestantes > 0:
            self.vitesse = self.iterationsRestantes/self.iterations * self.vitesseInitiale
            self.iterationsRestantes -= 1


    def calculDistanceFreinage(self : object) -> float:
        distance = 0
        if self.frein == 'Patin' and cst.temps == 'Sec':
            for i in range(len(cst.patinsS)):
                distance += cst.patinsS[i]*(self.vitesse**i) #Distance nécessaire pour
s'arrêter
        if self.frein == 'Disque' and cst.temps == 'Mouille':
            for i in range(len(cst.disquesM)):
                distance += cst.disquesM[i]*(self.vitesse**i)
        if self.frein == 'Disque' and cst.temps == 'Sec':
            for i in range(len(cst.disquesS)):
                distance += cst.disquesS[i]*(self.vitesse**i)
        if self.frein == 'Patin' and cst.temps == 'Mouille':
            for i in range(len(cst.patinsM)):
                distance += cst.patinsM[i]*(self.vitesse**i)
        return distance

#Affichage de l'échelle
can.create_line(10,10,10 + cst.pixel,10, arrow = BOTH)
can.create_text(10+cst.pixel/2,20, text = '1m')

def chercheCase(x : float, y : float) -> list:
    '''Fonction qui cherche la case cliquée à partir des coordonnées'''
```

```
    for i in range(hauteur):
        for j in range(cst.largeur):
            coordonnees = can.coords(grille[i][j])
            if coordonnees[0] <= x <= coordonnees[2] and coordonnees[1] <= y <= coordonnees[3]:
                return can.gettags(grille[i][j])[1].replace('case','').split(',')
    return [-1,-1]

def clavier(event : Event) -> None:
    '''Fonction pour interagir avec le clavier'''
    touche = event.keysym
    if touche == 'ampersand':
        changeGrille()
    if touche == 'space':
        mouvementALL()

def changeGrille() -> None:
    '''Fonction qui affiche/dissimule la grille'''
    bordure = can.itemcget('grille', 'outline')
    if bordure == '': #On change la bordure en fonction
        can.itemconfig('grille',outline = 'black')
    else:
        can.itemconfig('grille',outline = '')

#Création de la grille
grille = [[can.create_rectangle(x*cst.case,y*cst.case,(x+1)*cst.case,(y+1)*cst.case,
tag = ('grille','case' + str(y) + ',' + str(x))) for x in range(cst.largeur)] for y in range(hauteur)]
#En créant chaque case, on la munit d'un tag général 'grille' et d'un tag qui identifie sa position dans la liste

def valider(case : list, vitesse : str, frein : str) -> None:
    '''Fonction qui traite les paramètres renseignés dans la fenêtre'''
    global listeVelo, fenVelo
    if frein != '' and vitesse.isdigit():
        listeVelo.append(cycliste(int(vitesse),frein,case))
        fenVelo.destroy()
    else:
        showerror('Erreur', 'Les paramètres renseignés ne sont pas valides.')

def creationPossible(cx : str, cy : str, ignore = [], renvoie = False) -> bool:
    '''Fonction qui indique s'il est possible de créer un vélo à la case renseigné'''
    global listeVelo
    if int(cx) + cst.nombreCase[0] > hauteur or int(cy) + cst.nombreCase[1] > cst.largeur:
        if renvoie:
            return False, None
        return False
    #On récupère les coordonnées des quatres coins du vélo que l'on veut ajouter
    tagCaseHautGauche = 'case' + cx +',' + cy
    tagCaseBasDroit = 'case' + str(int(cx)+cst.nombreCase[0]-1) +',' + str(int(cy)+cst.nombreCase[1]-1)
    absg = can.coords(tagCaseHautGauche)[0] #Abscisse du bord gauche
    absd = can.coords(tagCaseBasDroit)[2] #Abscisse du bord droit
    ordh = can.coords(tagCaseHautGauche)[1] #Ordonnée du bord haut
```

```
200     ordb = can.coords(tagCaseBasDroit)[3] #Ordonnée du bord bas
201     coins = [[absg,ordh],[absd,ordh],[absd,ordb],[absg,ordb]]
202     #On regarde s'il y a collision
203     for velo in listeVelo:
204         if velo not in ignore:
205             gx, gy, dx, dy = can.coords(velo.id)
206             if (gx == absg or dx == absd) and (gy < ordh < dy or gy < ordb < dy):
207                 if renvoie:
208                     return False, velo
209                 return False
210             if (gy == ordh or dy == ordb) and (gx < absg < dx or gx < absd < dx):
211                 if renvoie:
212                     return False, velo
213                 return False
214             if gx == absg and gy == ordh:
215                 if renvoie:
216                     return False, velo
217                 return False
218             for coin in coins:
219                 if gx < coin[0] < dx and gy < coin[1] < dy:
220                     if renvoie:
221                         return False, velo
222                     return False
223     if renvoie:
224         return True, None
225     return True
226 
227 def menuModification(velo : cycliste, vitesse : str, frein : str) -> None:
228     '''Fonction qui modifie les paramètres renseignés dans la fenêtre'''
229     global fenVelo
230     if frein != '' and vitesse.isdigit():
231         velo.vitesse = int(vitesse)
232         if not velo.freinageEnCours:
233             velo.vitesseInitiale = int(vitesse)
234         velo.frein = frein
235         can.itemconfig(velo.id, outline = 'red')
236         fenVelo.destroy()
237     else:
238         showerror('Erreur', 'Les paramètres renseignés ne sont pas valides.')
239 
240 def fermetureFenVelo() -> None:
241     '''Fonction qui déselectionne le vélo quand la fenêtre de modification est fermée'''
242     global fenVelo
243     deselectionnerVelo(None)
244     fenVelo.destroy()
245 
246 def deselectionnerVelo(id : int) -> None:
247     '''Fonction qui remet en rouge tous les vélos de la liste sauf celui en argument'''
248     global listeVelo
249     for velo in listeVelo:
250         if velo.id != id:
251             can.itemconfig(velo.id, outline = 'red')
252 
253 def creationFenetre(velo : cycliste, case = []) -> None:
```

```
    global fenVelo
    #Ensuite on crée un fenêtre temporaire pour que l'utilisateur rentre les attributs du vélo qu'il veut créer
    fenVelo = Toplevel(fen)
    #Variables qui contiendrons les différents paramètres renseignés
    vitesse = StringVar()
    frein = StringVar()
    freins = ['Patin', 'Disque']
    if velo != None: #Si le vélo est déjà crée et que l'on veut le modifier, on met ses attributs
        frein.set(velo.frein)
        vitesse.set(velo.vitesse)
    else:
        vitesse.set(0)
        frein.set('Patin')
    #Création des widgets de la fenêtre
    labelVitesse = Label(fenVelo, text='Vitesse (km/h)')
    spinboxVitesse = Spinbox(fenVelo, from_ = 0, to_ = 100, increment = 1, textvariable = vitesse, width=5)
    labelFrein = Label(fenVelo, text = 'Choix du frein')
    comboboxFrein = ttk.Combobox(fenVelo,values = freins,textvariable = frein,width=7)
    if velo != None:
        boutonChute = Button(fenVelo, text = 'Faire chuter', command = velo.chuter)
        boutonSupprimer = Button(fenVelo,text = 'Supprimer', command = velo.supprimer)
        boutonChute.pack()
        boutonSupprimer.pack()
        boutonValider = Button(fenVelo, text='Sauvegarder', command = lambda:menuModification(velo, vitesse.get(), frein.get()))
        #On lie les quatres flèches du clavier pour pouvoir déplacer le vélo
        fen.bind('<Left>', lambda event:velo.decaleGauche())
        fen.bind('<Right>', lambda event:velo.decaleDroite())
        fen.bind('<Up>', lambda event:velo.decaleHaut())
        fen.bind('<Down>',lambda event:velo.decaleBas())
    else:
        boutonValider = Button(fenVelo, text='Valider', command = lambda:valider(case, vitesse.get(), frein.get()))
    #Affichage des widgets
    labelVitesse.pack(pady=5)
    spinboxVitesse.pack(pady=5)
    labelFrein.pack(pady=5)
    comboboxFrein.pack(pady=5)
    boutonValider.pack(pady=5)
    fenVelo.protocol("WM_DELETE_WINDOW", fermetureFenVelo)

def trouveVeloListe(id : int) -> cycliste:
    '''Fonction qui renvoie un vélo de la liste grâce à ses coordonnées en argument'''
    global listeVelo
    for velo in listeVelo:
        if velo.id == id:
            return velo
    raise IndexError

def fermeture() -> None:
```

```
    '''Fonction qui ferme la fenêtre après avoir modifier le fichier de sauvegarde des vélos'''
    global listeVelo
    lignesVelo = ''
    for velo in listeVelo:
        lignesVelo+=str(velo.case[0])+' '+str(velo.case[1])+' '+str(velo.vitesseInitiale)+' '+velo.frein+' '+str(velo.chute)+' \n'
        #On réécrit dans le fichier la nouvelle liste de vélo
    fichierVelo = open('flotte.txt','w') #On ouvre le fichier en modification
    fichierVelo.seek(0) #On supprime le contenu du fichier
    fichierVelo.write(lignesVelo)
    fichierVelo.close() #On ferme le fichier
    fen.destroy() #On ferme ensuite la fenêtre

def clicGauche(event : Event) -> None:
    '''Fonction qui gère un clic gauche sur le Canvas'''
    global frein, vitesse, fenVelo,listeVelo
    #Récupération des coordonnées du clic
    x,y = event.x, event.y
    #Modifier les attributs d'un vélo
    objs = can.find_closest(x,y)
    tags = can.gettags(objs)
    if fenVelo is not None:
        fenVelo.destroy()
    for i in range(len(tags)):
        if tags[i] == 'velo':
            if can.itemcget(objs[i],'outline') == 'red':
                can.itemconfig(objs[i],outline = 'green')
                creationFenetre(trouveVeloListe(objs[i]))
            else:
                can.itemconfig(objs[i],outline = 'red')
            for velo in listeVelo:
                if can.itemcget(velo.id,'outline') == 'green':
                    deselectionnerVelo(objs[i])
                    if velo.id != objs[i]:
                        can.itemconfig(velo.id,outline = 'red')
                        break
    #Créer un vélo à partir d'un clic sur la grille
    #Récupération de la case cliquée
    cx,cy = chercheCase(x,y)
    if [cx,cy] != [-1,-1] and creationPossible(cx,cy):
        #On appelle la fonction qui crée la fenêtre avec en argument la fonction qui permet de créer un vélo quand on valide
        deselectionnerVelo(None)
        creationFenetre(None, [int(cx),int(cy)])

#Ajout dans la liste des vélos ceux déjà crées
for ligne in lignesVelo:
    l = ligne.split(' ')
    listeVelo.append(cycliste(int(l[2]),l[3],[int(l[0]),int(l[1])],l[4]=="True"))

def plusADroite() -> list:
    '''Fonction qui renvoie la liste des vélos dans l'ordre du plus à droite en premier jusqu'au plus à gauche en dernier'''
```

```
352     global listeVelo
353     ordre = []
354     while len(ordre) != len(listeVelo):
355         max = None
356         for j in listeVelo:
357             if j not in ordre and (max is None or can.coords(j.id)[2] >= can.coords(max.id)
    [2]):
358                 max = j
359         ordre.append(max)
360     return ordre
361 
362 def decaleHaut(x : float, y : float, velo : cycliste, veloGenant = [], res = 0):# ->
    cycliste,int
363     '''Fonction qui renvoie le nombre de cases
364     à se décaler vers le haut pour être en sécurité'''
365     global listeVelo
366     for v in listeVelo:
367         if v!=velo and (v.vitesse < velo.vitesse or v.chute):
368             x1,y1,_,_ = can.coords(v.id)
369             if x1 >= x and (y1 == y or y1 == y + cst.case or y1 == y - cst.case):
370                 if v not in veloGenant:
371                     return decaleHaut(x,y-cst.case,velo,veloGenant+[v],res+1)
372                 return decaleHaut(x,y-cst.case,velo,veloGenant,res+1)
373     return veloGenant,res
374 
375 def decaleBas(x : float, y : float, velo : cycliste, veloGenant = [], res = 0):
376     '''Fonction qui renvoie le nombre de cases à se décaler vers le bas pour être en
    sécurité'''
377     global listeVelo
378     for v in listeVelo:
379         if v!=velo and (v.vitesse < velo.vitesse or v.chute):
380             x1,y1,_,_ = can.coords(v.id)
381             if x1 >= x and (y1 == y or y1 == y + cst.case or y1 == y - cst.case):
382                 if v not in veloGenant:
383                     return decaleBas(x,y+cst.case,velo,veloGenant+[v],res+1)
384                 return decaleBas(x,y+cst.case,velo,veloGenant,res+1)
385     return veloGenant,res
386 
387 def collision(x : float,y : float, velo : cycliste, renvoie = False) -> bool:
388     global listeVelo
389     for v in listeVelo:
390         if v != velo:
391             x1,y1,x2,y2 = can.coords(v.id)
392             if x < x1 < x+cst.case*cst.nombreCase[1] and (y1 == y or y1 == y - cst.case or
    y1 == y + cst.case):
393                 if renvoie:
394                     return False,v
395                 return False
396             if x < x2 < x+cst.case*cst.nombreCase[1] and (y1 == y or y1 == y - cst.case or
    y1 == y + cst.case):
397                 if renvoie:
398                     return False,v
399                 return False
400     if renvoie:
```

```
401         return True,-1
402     return True
403 
404 def mouvementALL() -> None:
405     '''Fonction qui fait avancer les vélos'''
406     global listeVelo
407     for velo in plusADroite():
408         mouvement(velo)
409 
410 def deplacementPossible(velo : cycliste, case : int, deplacement : float, signe : int):
411     #On vérifie que si le vélo doit se déplacer de case cases, il ne risque pas de collision
412     global listeVelo
413     X1,Y1,X2,Y2 = can.coords(velo.id)
414     X1Temp = X1
415     for i in range(1,case+1):
416         X1 += deplacement
417         X2 += deplacement
418         Y1 += signe * cst.case
419         Y2 += signe * cst.case
420         if Y2 > cst.case*hauteur or Y1 < 0:
421             return False
422         for v in listeVelo:
423             x1,y1,x2,_ = can.coords(v.id)
424             dejaAvance = 1
425             if x1 < X1Temp: #Si le vélo est avant, il n'a pas encore bougé
426                 dejaAvance = 0
427             x1 +=convertirVitesse(v.vitesse)*cst.dt*cst.pixel*(i-dejaAvance)
428             x2 +=convertirVitesse(v.vitesse)*cst.dt*cst.pixel*(i-dejaAvance)
429             if X1 <= x1 <= X2 and (Y1 == y1 or Y1 == y1 - cst.case or Y1 == y1 + cst.case):
430                 return False
431             if X1 <= x2 <= X2 and (Y1 == y1 or Y1 == y1 - cst.case or Y1 == y1 + cst.case):
432                 return False
433     return True
434 
435 def mouvement(velo : cycliste) -> None:
436     global listeVelo
437     if velo.vitesse < 0:
438         velo.vitesse = 0
439         velo.freinageEnCours = False
440     if not velo.chute:
441         deplacementMetre = convertirVitesse(velo.vitesse) * cst.dt # Nombre de mètres à se
    déplacer
442         deplacementPixel = deplacementMetre * cst.pixel #On convertit cette distance en
    pixels
443         deplacementY = False #Variable qui indique si le vélo s'est décalé
444         #Variable qui permettra de déduire    la distance que peut avancer le vélo : s'il
    doit se décaler -> il fera moins de distance vers l'avant
445         deplacementRestant2 = deplacementPixel**2 - cst.case**2 #On détérmine la distance que
    peut encore faire le vélo s'il doit se décaler
446         #On considère qu'à partir d'une certaine vitesse trop faible, le vélo ne peut que
    avancer et pas se décaler
447         x1, y1, x2, y2 = can.coords(velo.id)
448         _,caseHaut = decaleHaut(x2,y1,velo)
449         _,caseBas = decaleBas(x2,y1,velo)
```

```
        if caseBas == caseHaut == 0: #S'il n'y a pas besoin de se décaler
            can.move(velo.id,deplacementPixel,0) #On le fait avancer normalement
        else: #S'il y a besoin de se décaler
            if deplacementRestant2 > 0:
                deplacementPossibleHaut = deplacementPossible(velo,caseHaut,sqrt(deplacementRestant2),-1)
                deplacementPossibleBas = deplacementPossible(velo,caseBas,sqrt(deplacementRestant2),1)
                if caseHaut >= caseBas: #On teste différents cas
                    if y2 < hauteur*cst.case and collision(x1+sqrt(deplacementRestant2),y1+cst.case,velo) and deplacementPossibleBas:
                        deplacementY = True
                        can.move(velo.id,0,cst.case)
                    elif y1 > 0 and collision(x1+sqrt(deplacementRestant2),y1-cst.case,velo) and deplacementPossibleHaut:
                        deplacementY = True
                        can.move(velo.id,0,-cst.case)
                else:
                    if collision(x1+sqrt(deplacementRestant2),y1-cst.case,velo) and y1 > 0 and y1-cst.case*caseHaut > 0 and deplacementPossibleHaut:
                        deplacementY = True
                        can.move(velo.id,0,-cst.case)
                    elif collision(x1+sqrt(deplacementRestant2),y1+cst.case,velo) and y2 < hauteur*cst.case and deplacementPossibleBas:
                        deplacementY = True
                        can.move(velo.id,0,cst.case)
            if deplacementY: #Si on s'est décalé
                can.move(velo.id,sqrt(deplacementRestant2),0) #On bouge de la distance restante
            else: #Si on ne peut pas
                can.move(velo.id,deplacementPixel,0) #On avance quoi qu'il en soit
                velo.freinage() #Et on freine
        if deplacementY: # On regarde si le vélo a freiné ou s'il a fini
            velo.freinageEnCours = False
        #On détecte si le vélo en a percuté un autre
        percute, v = collision(can.coords(velo.id)[0],can.coords(velo.id)[1],velo,True)
        if not percute and v is not None: #On les fait chuter s'ils se percutent
            velo.chuter()
            v.chuter()

def supprimerVelos() -> None:
    global listeVelo
    can.delete('velo')
    listeVelo = []

def genererVelos(n : int, vitesseMin : float, vitesseMax : float, freinDefaut : str) -> None: #Fonction qui créer un certain nombre de vélos
    global listeVelo
    for _ in range(n):
        vitesse = randint(vitesseMin,vitesseMax) # On génère une vitesse
        if resultatObstacle:
            case = [randint(0,hauteur-cst.nombreCase[0]),randint(0,(cst.largeur-cst.nombreCase[1])*2//3)] # On génère une case
        else:
```

```
            case = [randint(0,hauteur-cst.nombreCase[0]),randint(0,cst.largeur-cst.nombreCase[1])] # On génère une case
        if creationPossible(str(case[0]),str(case[1])): # Si la case n'est pas déjà occupée
            listeVelo.append(cycliste(vitesse,freinDefaut,case)) # On créer le vélo
    if resultatLargeur or resultatFrein:
        i = randint(0,len(listeVelo)-1)
        listeVelo[i].chuter()

def compteChute() -> int:
    global listeVelo
    res=0
    for v in listeVelo:
        if v.chute:
            res+=1
    return res

#Interaction avec le clavier et la souris
can.bind('<Button-1>',clicGauche)
fen.bind('<Key>',clavier)

#On appelle la fonction qui sauvegarde la liste de vélo quand la fenêtre est fermée
fen.protocol("WM_DELETE_WINDOW", fermeture)

#Bouton pour continuer la simulation
continuer = Button(fen,text = 'Continuer',command=mouvementALL)
continuer.pack(pady = 5)

#Bouton pour supprimer tous les vélos
supprimerTout = Button(fen,text = 'Supprimer tout',command=supprimerVelos)
supprimerTout.pack(pady = 5)

#Entrée pour renseigner le nombre de vélos à générer
genereVar = IntVar()
entreeGenerer = Entry(fen,textvariable=genereVar)
entreeGenerer.pack(pady=5)

#Bouton pour générer des vélos
generer = Button(fen, text = 'Générer les vélos', command = lambda:genererVelos(genereVar.get()))
generer.pack(pady = 5)

def ecriture(liste : list) -> None:
    lignesVelo = ''
    for velo in liste:
        lignesVelo+=str(velo.case[0])+' '+str(velo.case[1])+' '+str(velo.vitesseInitiale)+' '+velo.frein+' '+str(velo.chute)+' \n'
    fichierVelo = open('flotte.txt','w') #On ouvre le fichier en modification
    fichierVelo.seek(0) #On supprime le contenu du fichier
    fichierVelo.write(lignesVelo)
    fichierVelo.close() #On ferme le fichier

def compteChute():
    global listeVelo
    chute = 0
```

```
    for velo in listeVelo:
        if velo.chute:
            chute += 1
    return chute

if resultatLargeur:
    if hauteur == 22:
        genererVelos(100, 25, 35, "Patin")
        ecriture(listeVelo)
    can.move('velo', 0, (hauteur-22)*cst.case//2)

if resultatObstacle:
    if not obstacle:
        genererVelos(55, 30, 35, "Disque")
        ecriture(listeVelo)
    else:
        obstacle1 = cycliste(0, 'Patin',[9,102], True)
        listeVelo.append(obstacle1)
        obstacle2 = cycliste(0, 'Patin',[11,102], True)
        listeVelo.append(obstacle2)

if resultatFrein:
    if proportion == 0:
        genererVelos(100, 30, 35, "Patin")
        ecriture(listeVelo)
    for j in range((len(listeVelo) - 1) * proportion//100):
        listeVelo[j].frein = 'Disque'

if resultatFrein or resultatLargeur or resultatObstacle:
    for _ in range(20):
        mouvementALL()
    print(compteChute())
else:
    fen.mainloop()
```

resultat.py

```python
#Importation du module pour exécuter des fichiers Python
import subprocess

#Variables
case = 11
pixel = 50

#Obtention des résultats
resultatLargeur = False
resultatObstacle = False
resultatFrein = True
nombreSimulations = 2

def changePourcentage(p : int) -> None:
    fichier = open("proportion.txt","w")
    fichier.seek(0)
    fichier.write(str(p))
    fichier.close()

def changeHauteur(n : int) -> None:
    fichier = open("hauteur.txt","w")
    fichier.seek(0)
    fichier.write(str(n))
    fichier.close()

def changeObstacle(b : bool) -> None:
    fichier = open("obstacle.txt","w")
    fichier.seek(0)
    fichier.write(str(b))
    fichier.close()

def moyenne(l : list) -> float:
    return sum(l)/len(l)

if resultatLargeur:
    resultats = {5 : [], 6 : [], 7 : [], 8 : []}
    for _ in range(nombreSimulations):
        for largeur in [5, 6, 7, 8]:
            changeHauteur(largeur*pixel//case)
            resultat = subprocess.run(['python', 'main.py'], capture_output=True, text=True)
            resultats[largeur].append(int(resultat.stdout))
    for largeur in [5, 6, 7, 8]:
        print(largeur, moyenne(resultats[largeur]))

if resultatObstacle:
    resultats = {True : [], False : []}
    for _ in range(nombreSimulations):
        for obstacle in [False, True]:
            changeObstacle(obstacle)
            resultat = subprocess.run(['python', 'main.py'], capture_output=True, text=True)
            resultats[obstacle].append(int(resultat.stdout))
```

```
    for obstacle in [False, True]:
        print(obstacle, moyenne(resultats[obstacle]))

if resultatFrein:
    resultats = {0 : [], 25 : [], 50 : [], 75 : [], 100 : []}
    for _ in range(nombreSimulations):
        for pourcentage in [0, 25, 50, 75, 100]:
            changePourcentage(pourcentage)
            resultat = subprocess.run(['python', 'main.py'], capture_output=True, text=True)
            resultats[pourcentage].append(int(resultat.stdout))
    for pourcentage in [0, 25, 50, 75, 100]:
        print(pourcentage, moyenne(resultats[pourcentage]))

```