Aufgabe 1: Schiebeparkplatz

Team-ID: 01048

Team: Lorian

Bearbeiter/-innen dieser Aufgabe:
Lorian Linnertz

17. November 2021

Inhaltsverzeichnis

Lösungsidee

Meine Lösungsidee war es, einmal die horizontalen Autos nach links und danach nach rechts fahren zu lassen, und am Ende zu schauen, welcher der beiden Wege kürzer ist. Um die Auto zu verschieben muss man vor jeder Verschiebung überprüfen, ob man gegen die Wand fahren würde oder ob man mit einem anderen Auto kollidieren würde. Falls diese beiden Punkte nicht zu treffen, so kann man das Auto ohne Probleme verschieben, würde man gegen ein anderes Auto treffen, so müsste man dieses ebenfalls verschieben und wenn man gegen die Wand fährt, so sind ebenfalls alle vorherigen Verschiebungen ungültig.

Mann muss auch noch hinzufügen, dass entweder alle Autos in die eine oder die andere Richtung fahren. Das heißt auch, dass man später beim Ausgeben der Resultate, nur die gesammten Bewegungen eines horizontalen Autos in eine Richtung ausgeben muss, wobei zu sagen ist, dass dies entgegen der Aufzählrichtung passiert. (d.h. wenn man die Bewegung von den Autos aufzählen will die von links gefahren kommen, so mann man von rechts nach links die Autos aufzählen. Bei rechts ist es genau anders rum)

Umsetzung

Für die Umsetzung, habe ich einfach die in der Lösungsidee blau markierten Schritt in einer Rekursiven Funktion zusammen gefasst. Bedeutet, dass alle Autos erst fahren, wenn sicher ist, dass keines gegen die Wand fährt. Ebenfalls habe ich das Programm in zwei Klassen aufgeteilt. Eine

Klasse mit den vertikalen Autos und eine mit den horizontalen. In der Klasse der Vertikalen Autos, was die Hauptklasse darstellt, befindet sich die rekursive Methode, was das Herzstück diese Programs darstellt und in der Klasse der horizontalen Autos befindet sich die Methode zur Überprüfung der Begrenzungen. Die rekursive Funktion verteilt sich auf zwei Methoden, die moveSript() und die checkCollision() Methoden, diese Rufen sich gegenseitig immer wieder auf und geben sich ständig True, False Werte hin und her. Im ersten Durchgang, wir die Verschiebung der linken Autos berechnet, und anschließend werden die Posiotionen der horizontalen Autos mit der resetCars() Methode wieder in den Anfangszustand gesetzt zur berechnung der Bewegung der rechten Bewegungen. Die Durchgänge und sämtliche anderen Methoden werden von der Hauptmethode controller() kontrolliert

Beispiele

########## PARKPLATZ0.TXT ##########

A:
B:
C: H 1 rechts
D: H 1 links
E:
F: H 1 links, I 2 links
G: I 1 links

########## PARKPLATZ1.TXT ##########

A:
B: P 1 rechts, O 1 rechts
C: O 1 links
D: P 1 rechts
E: O 1 links, P 1 links
F:
G: Q 1 rechts
H: Q 1 links
I:
J:
K: R 1 rechts
L: R 1 links
M:
N:

########## PARKPLATZ2.TXT ##########

A:
B:
C: O 1 rechts
D: O 1 links
E:
F: O 1 links, P 2 links
G: P 1 links
H: R 1 rechts, Q 1 rechts
I: P 1 links, Q 1 links
J: R 1 rechts
K: P 1 links, Q 1 links, R 1 links
L:
M: P 1 links, Q 1 links, R 1 links, S 2 links
N: S 1 links

########## PARKPLATZ3.TXT ##########

A:
B: O 1 rechts
C: O 1 links
D:
E: P 1 rechts
F: P 1 links
G:
H:
I: Q 2 links
J: Q 1 links
K: Q 2 links, R 2 links
L: Q 1 links, R 1 links
M: Q 2 links, R 2 links, S 2 links
N: Q 1 links, R 1 links, S 1 links

########## PARKPLATZ4.TXT ##########

A: R 1 rechts, Q 1 rechts
B: R 2 rechts, Q 2 rechts
C: R 1 rechts
D: R 2 rechts
E:
F:
G: S 1 rechts
H: S 1 links
I:
J:
K: T 1 rechts
L: T 1 links
M:
N: U 1 rechts
O: U 1 links
P:

########## PARKPLATZ5.TXT ##########

A:
B:
C: P 2 links
D: P 1 links
E: Q 1 rechts
F: P 1 links, Q 1 links
G:
H:
I: R 1 rechts
J: R 1 links
K:
L:
M: S 1 rechts
N: S 1 links
O:

Quellcode

```
class VerticalCar:

    def __init__(self,parkID,parkRange, ListOf_HorizontalCars):#die parkRange ist eine Liste mit dem kleinsten ord Wert und dem grössten der VerticalCars und die ListHorizontalCars ist eine Liste mit jeweils einer Liste welche die Informationen zu den Horzontal Cars het mit dem Format [ID,Postition]

        self.parkID = parkID #die parkID ist der Standplatz

        self.carID = parkID + 65
```

```
        self.parkRange = parkRange
        self.horizontalCars = VerticalCar.createHorizontalCars(self, ListOf_HorizontalCars)
        self.counterList = [0,0] #[0] steht für right und [1] für left
        self.controlBool = [True,True]#[0] steht für right und [1] für left, diese Variable wird auf False gesetzt sobald eine Bewegung nicht mehr mölich ist
        self.notFreeDrive = VerticalCar.checkExit(self)[0] #diese Variable gibt an ob das Auto am Anfang freie fagrt hatte
        VerticalCar.controller(self)

    def resetCars(self):
        for car in self.horizontalCars:
            car.position = car.startPosition

    def checkExit(self):
        for car in self.horizontalCars:
            if car.position[0] == self.parkID or car.position[1] == self.parkID:
                return [True,car] # "True" besagt, dass die Ausfahrt versperrt ist und das zweite Element gibt zurück von was
        return [False,None] #"False" besagt, dass die Ausfahrt nicht verspert ist, und gibt ebenfalls None zurück, das kein Auto die Ausfahrt versperrt

    def controller(self): #Diese Funktion soll später alle anderen Ansteuern
        if self.notFreeDrive and VerticalCar.checkExit(self)[1].checkBorder(-1): #Falls keine Freifahrt ist, wird versucht, das Auto nach links zu schieben
            for _ in range(2):
                VerticalCar.moveScript(self,VerticalCar.checkExit(self)[1],-1)
                if self.controlBool == False: #Falls controlBool False ist, so wird die schleife beendet und der Counter zurückgesetzt
                    self.counterList[0] = -1 #
                    break
                elif VerticalCar.checkExit(self)[0] == False:
                    break
            if VerticalCar.checkExit(self)[0] == True:
                self.counterList[0] = -1

        elif self.notFreeDrive and VerticalCar.checkExit(self)[1].checkBorder(-1) == False:
            self.counterList[0] = -1

        VerticalCar.resetCars(self)
        if self.notFreeDrive and VerticalCar.checkExit(self)[1].checkBorder(1): #Falls keine Freifahrt ist, wird versucht, das Auto nach rechts zu schieben
            for _ in range(2):
```

```
            VerticalCar.moveScript(self,VerticalCar.checkExit(self)[1],1)
            if self.controlBool == False: #Falls controlBool False ist, so wird die schleife beendet und der Counter zurückgesetzt
                self.counterList[1] = -1 #
                break
            elif VerticalCar.checkExit(self)[0] == False:
                break
        if VerticalCar.checkExit(self)[0] == True:
            self.counterList[1] = -1
    elif self.notFreeDrive and VerticalCar.checkExit(self)[1].checkBorder(1) == False:
        self.counterList[1] = -1


def moveScript(self,car,movementDircection):
    if VerticalCar.checkCollision(self,car,movementDircection) and car.checkBorder(movementDircection):
        car.move(movementDircection) #Beweget das "car"
        if movementDircection == -1: #Dieses If-Statment soll alle Schritte zählen
            self.counterList[0] +=1
        elif movementDircection == 1:
            self.counterList[1] +=1
        return True
    else:
        if movementDircection == -1: #Dieses If-Statment soll alle Schritte zählen
            self.controlBool[0] = False
        elif movementDircection == 1:
            self.controlBool[1] == False
        return False



def checkCollision(self,car,movementDircection): #Überprüft, ob bei dem nächsten Schritt von Car, das Car gegen ein anderes Auto renn und verschiebt das betroffene Auto wenn möglich
    for i in self.horizontalCars:
        if movementDircection == -1 and (car.position[0]-1) in i.position: #überprüft ob car in das Auto "i" gestoßen ist.
            if i.checkBorder(-1): #überprüft ob das Auto im nächsten Schritt gegen die Begrenzung fährt. Bedeutet False: würde gegen die Begrenzung fahren; True: Würde nicht gegen die Begrenzung fahren
                return VerticalCar.moveScript(self,i,-1) #Falls beides stimmt, so gebe Folgende Methode zurück "Rekursive Funktion"
            else:
                return False #Gibt False zurück falls das Auto gegen eine Wand fährt
        elif movementDircection == 1 and (car.position[1]+1) in i.position: #genau das gleiche wie vorhin nur für die andere Richtung
            if i.checkBorder(1):
                return VerticalCar.moveScript(self,i,1)
```

```
            else:
                return False #Gibt False zurück falls das Auto gegen eine Wand fährt
        return True #Gibt zurück, dass keine Kollision stattgefundenhat.


    def createHorizontalCars(self, ListOf_HorizontalCars):
        horizontalCarsList = []
        for car in ListOf_HorizontalCars:
            horizontalCarsList.append(HorizontalCar(car[0],car[1],self.parkRange))
        return horizontalCarsList


class HorizontalCar:
    def __init__(self,carID,PostitionA,parkRange): #die Car ID ist der (Ord Wert - 65) seines Buchstabens
        self.carID = carID
        self.startPosition = [PostitionA, PostitionA+1] #[erste Koordinate, zweite Koordinate]
        self.movementCounter = [0,0] #Die linke Spalte ist für die Bewegungen nach links und die rechte für die Bewegungen nach rechts
        self.position = []
        self.position.extend(tuple(self.startPosition))
        self.parkRange = parkRange


    def move(self,movementDircection): #Diese Funktion, soll das Auto bewegen und die Bewegungen in jede Richtung zählen
        self.position[0] += movementDircection
        self.position[1] += movementDircection
        if movementDircection == -1:
            self.movementCounter[0] += 1
        elif movementDircection == 1:
            self.movementCounter[1] += 1


    def checkBorder(self,movementDircection): #Diese Funktion überprüft ob der NÄCHSTE Schritt noch in der Deffinitionsmenge ist, diese lautet [0,self.parkRange]
        if self.position[0] <= 0 and movementDircection == -1: #Falls jetzt noch eine Bewegung ausgeführt werden sollte, so rennt das Auto in die Wand
            return False
        elif self.position[1] >= self.parkRange and movementDircection == 1:
            return False
        else:
            return True #Gibt True zurück falls der nächste Schritt möglich ist und False falls nicht
```