Betrunkener

Dickbauer Y., Moser P., Perner M.

PS Computergestützte Modellierung, WS 2016/17

December 15, 2016

Outline

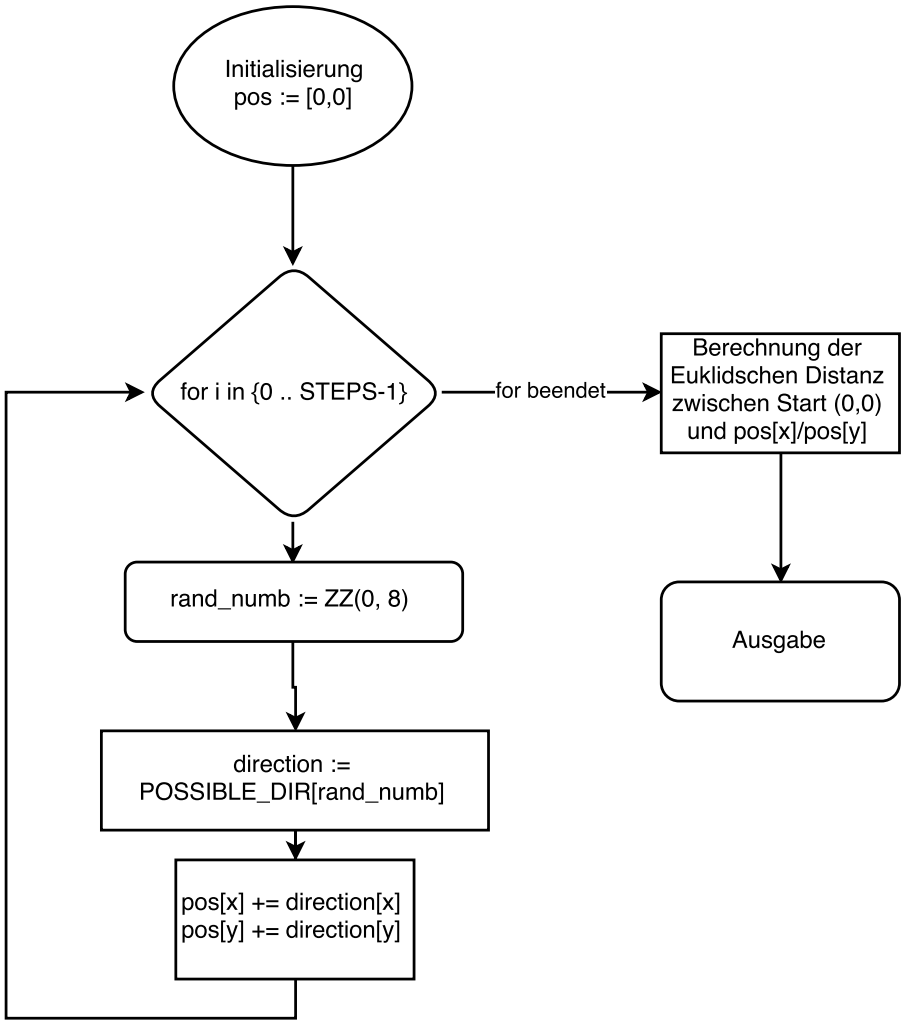
Aufgabenstellung

In der Mitte eines großen Platzes steht ein Betrunkener an einem Baum gelehnt. Er entschließt sich zum Gehen, ohne ein bestimmtes Ziel anzustreben. Folgende Schritte sind möglich (in Längeneinheiten):

Richtung	N	O	S	W	NW	NO	SO	SW
Wahrscheinlichkeit	0/1	1/0	0/-1	-1/0	-1/1	1/1	1/-1	-1/-1

- Eingabe: Anzahl an Schritten
- Output: Entfernung vom Ausgangspunkt nach n Schritten

Flow Chart


Initialisierung
pos := [0,0]
for i in {0 .. STEPS-1}
for beendet
Berechnung der Euklidschen Distanz zwischen Start (0,0) und pos[x]/pos[y]
rand_numb := ZZ(0, 8)
Ausgabe
direction := POSSIBLE_DIR[rand_numb]
pos[x] += direction[x]
pos[y] += direction[y]

Main Funktion - Programmeinstieg

```
1   POSSIBLE_DIRECTIONS = ( (0, 1), (1, 0), (0, -1), (-1, 0), (-1, 1), (1, 1), (1,-1
2   START_POSITION = (0, 0)
3
4   def main():
5       # user input
6       (number_of_steps, ) = user_input((
7           ('Number␣of␣steps', int, 2000), ), DEBUG)
8
9       pos = list(START_POSITION)
10      for i in range(number_of_steps):
11          # get a random direction
12          rand_numb = int(random_number_from_interval(0, len(POSSIBLE_DIRECTIONS)))
13          direction = POSSIBLE_DIRECTIONS[rand_numb]
14          # update current position
15          pos[0] += direction[0]
16          pos[1] += direction[1]
17
18      distance = euclidean_distance(START_POSITION, pos)
19      print('Aktueller␣Punkt␣({},␣{})␣->␣{:.2f}␣EH␣Entfernung␣zum␣Ausgangspunkt.'.
20          pos[0], pos[1], distance))
```

Funktion euclidean_distance(p1, p2)

- Diese Funktion verlangt zwei Punkte (x1, y1) (x2, y2) als Eingabeparameter
- Gibt die eukliedsche Distanz zurück

```
1   def euclidean_distance(point_1, point_2):
2       """
3           Calculates the euclidean distance between two points
4
5           point_1: a tuple of (x,y) values
6           point_2: a tuple of (x,y) values
7       """
8       delta_x = point_2[0] - point_1[0]
9       delta_y = point_2[1] - point_1[1]
10      return (delta_x ** 2 + delta_y ** 2) ** 0.5
```

Funktion random_number_from_interval(..)

- Diese Funktion verlangt zwei Eingabeparameter *lower* und *upper*
- Gibt eine (pseudo)Zufallszahl (*float*) im Intervall [*lower*, *upper*) zurück
- *random.random()* ist eine Funktion der Python Standardbibliothek, welche ein Zufallszahl (*float*) im Intervall [*lower*, *upper*) zurück gibt
- Mersenne Twister Methode wird als Generator der ZZ verwendet[1] [2]

```
1  def random_number_from_interval(lower, upper):
2      val = random.random()
3      return lower + (upper -lower) * val
```

[1] https://docs.python.org/3.5/library/random.html
[2] https://en.wikipedia.org/wiki/Mersenne_Twister

Funktion user_input(input_vars, [use_defaults])

- Diese Funktion verlang vom User die geforderten Eingabeparameter und gibt diese als von der Programmiererin gewünschten Datentyp wieder zurück
- Funktion verlangt als ersten Eingabeparameter die Liste *input_vars*
- Falls *use_defaults* == *True* wird der User nicht nach Eingabe gefragt (Dient zum Testen)
- Diese Liste besteht wiederrum aus Listen mit je Länge = 3:
  - 0: Text, welcher dem User ausgegeben wird
  - 1: Datentyp (int/float/str)
  - 2: Default value: Dieser Wert wird zurueckgegeben, falls *use_defaults* == *True*

```
1   x, y = user_input((
2       ('Geben Sie einen X Wert ein', int, 10),
3       ('Geben Sie einen Y Wert ein', int, 5), False):
```

Beispiel anhand fixer Zufallszahlen

- Annahme der Zufallszahlen wie folgt:

iteration	0	1	2	3	4	5-999
ZZ	0.05	0.21	0.20	0.22	0.09	0.09
rigged_dice	1	3	3	3	1	1

i := 0
- rigged_dice != 3 $\Rightarrow$ subsequent = 0, count = 0

i := 1
- rigged_dice == 3 $\Rightarrow$ subsequent = 1, count = 0

i := 2
- rigged_dice == 3 $\Rightarrow$ subsequent = 2, count = 0

Beispiel anhand fixer Zufallszahlen

i := 3

- rigged_dice == 3 $\Rightarrow$ subsequent = 3 $\Rightarrow$ count = 1

i := 4

- rigged_dice != 3 $\Rightarrow$ subsequent = 0, count = 1

Nach 1000 Iteration ist count = 1, also genau 1x 333 hintereinander

Anhang: Modifikation des Source Codes um Demo Beispiel zu erhalten

```
1     # Fuege folgenden Code vor random_number_from_interval() in lib.py ein:
2   ZZ = [0.05, 0.21, 0.20, 0.22] + [0.01]*1000
3   i = -1
4   def my_rand():
5       global i
6       i += 1
7       return ZZ[i]
8   random.random = my_rand
```