

# Betrunkener

Dickbauer Y., Moser P., Perner M.

PS Computergestützte Modellierung, WS 2016/17

December 15, 2016

# Outline

- 1 Aufgabenstellung
- 2 Flow Chart
- 3 Flow Chart
- 4 Programmcode
  - Main Funktion
  - Verwendete Funktionen
- 5 Beispiel

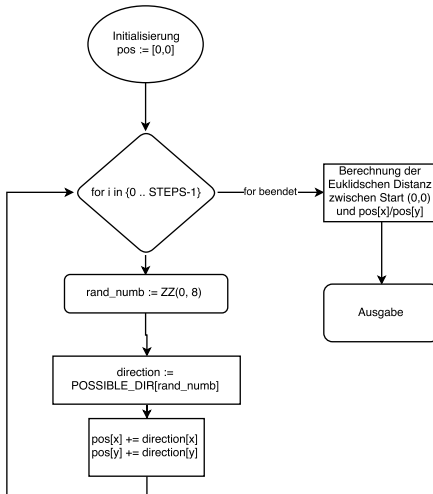
# Aufgabenstellung

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



# Main Funktion - Programmeinstieg

```
1 POSSIBLE_DIRECTIONS = ( (0, 1), (1, 0), (0, -1), (-1, 0), (-1, 1), (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_num = int(random_number_from_interval(0, len(POSSIBLE_DIRECTIONS)))
13        direction = POSSIBLE_DIRECTIONS[rand_num]
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.'.format(pos[0], pos[1], distance))
20    pos[0], pos[1], distance))
```



## Funktion `euclidean_distance(p1, p2)`

- Diese Funktion verlangt zwei Punkte  $(x_1, y_1)$   $(x_2, y_2)$  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
```



# 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

$i := 0$

- $\text{rigged\_dice} \neq 3 \Rightarrow \text{subsequent} = 0, \text{count} = 0$

$i := 1$

- $\text{rigged\_dice} == 3 \Rightarrow \text{subsequent} = 1, \text{count} = 0$

$i := 2$

- $\text{rigged\_dice} == 3 \Rightarrow \text{subsequent} = 2, \text{count} = 0$

# Beispiel anhand fixer Zufallszahlen

$i := 3$

- $\text{rigged\_dice} == 3 \Rightarrow \text{subsequent} = 3 \Rightarrow \text{count} = 1$

$i := 4$

- $\text{rigged\_dice} != 3 \Rightarrow \text{subsequent} = 0, \text{count} = 1$

Nach 1000 Iteration ist  $\text{count} = 1$ , 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
```

