

# KI HA 1

Tom Schmidt  
216204224

Stefan Poggenberg  
218100161

Samuel Schöpa  
216203821

Bjarne Hiller  
216203851

18. Mai 2018

## 1 Der Staubsauger-Agent

### 1.1 Implementierung

---

```
1 def ModelBasedVacuumAgent():
2     """An agent that keeps track of what locations are clean or
3         dirty."""
4     model = {}
5     direction = {(-1,0): 'Left', (1,0): 'Right', (0,-1): 'Up',
6                 (0,1): 'Down'}
7     last_location = None
8     last_move = None
9     moves = None
10
11     def program(percept):
12         nonlocal last_location
13         nonlocal last_move
14         nonlocal moves
15         (x, y), status = percept
16         model[(x, y)] = status
17
18         if not ((x, y) == last_location):
19             # new location: check for unknown neighbors
20             moves = [direction[(i, j)] for (i, j) in direction.
21                     keys() if not (x + i, y + j) in model]
22             last_location = (x, y)
23         else:
24             # last move had no effect: remove last_move
25             if last_move in moves:
26                 moves.remove(last_move)
```

```

25         if status == 'Dirty':
26             # Clean if location is dirty
27             return 'Suck'
28         elif len(moves) > 0:
29             # Into the unknown!
30             last_move = random.choice(moves)
31             return last_move
32         else:
33             return random.choice(['Right', 'Left', 'Up', 'Down'
34                                   ])
35     return Agent(program)

```

---

## 1.2 Optimaler Agent

Der implementierte Agent ist nicht optimal, da er für die Auswahl seiner Aktion nur die momentan benachbarten Felder beachtet. Sind alle benachbarten Felder schon entdeckt, wählt er eine zufällige Aktion. Ein optimaler Agent würde stattdessen berücksichtigen, in welche Richtung er sich bewegen müsste, um am schnellsten zu unbekannten Feldern zu kommen. Dies wäre dann aber schon ein Ziel-basierter Agent, da er zukünftige Beobachtungen und Aktionen berücksichtigt. Außerdem dürfte ein optimaler Agent keine Bewegungen mehr ausführen, wenn er sich sicher ist, dass er die gesamte Umgebung erkundet hat und alles sauber ist, da er so Performance verliert. Dies ist aber in der gegebenen Multi-Agent-Umgebung schwierig zu implementieren, da Felder nicht nur vom statischen Rand, sondern auch von anderen mobilen Agenten blockiert werden können.

## 2 Problemlösen durch Suchen

### 2.1 Suchalgorithmen

- a) i. Breitensuche: [Start, 7, Ziel]

Node	Queue
Start	1, 2, 7
1	2, 7, 4
2	7, 4, 5, 6
7	4, 5, 6, Ziel
4	5, 6, Ziel, 8
5	6, Ziel, 8
6	Ziel, 8, 3
Ziel	8

- ii. Tiefensuche: [Start, 1, 4, 8, Ziel]

Node	Stack
Start	7, 2, 1
1	7, 2, 4
4	7, 2, 8
8	7, 2, Ziel
Ziel	7, 2, 6

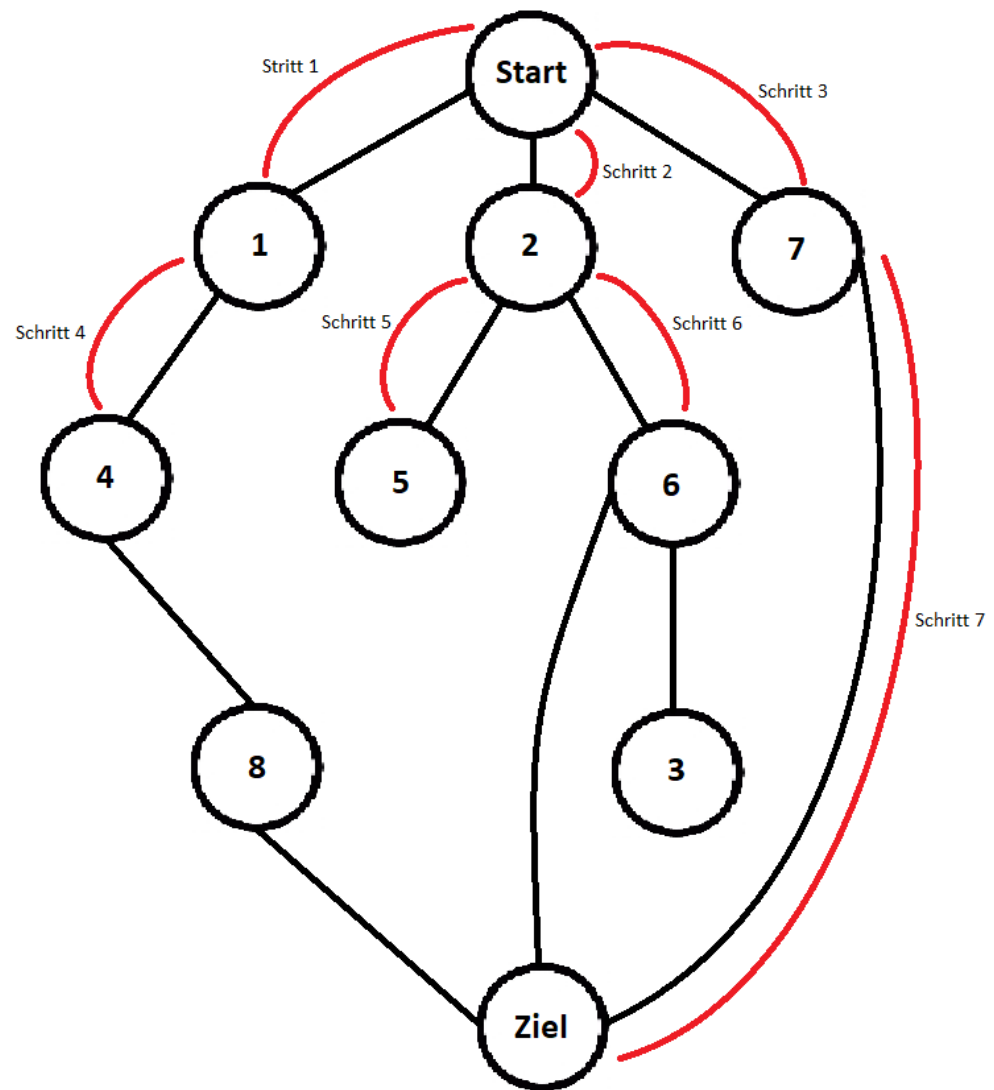
iii. A\*: [Start, 2, 6, Ziel]

Node	Discovered
Start	1 ( $85 + 272 = 357$ ), 2 ( $217 + 219 = 436$ ), 7 ( $173 + 383 = 556$ )
1	2 (436), 7 (556), 4 ( $85 + 80 + 253 = 418$ )
4	2 (436), 7 (556), 8 ( $85 + 80 + 250 + 57 = 472$ )
2	7 (556), 8 (472), 5 ( $217 + 186 + 318 = 721$ ), 6 ( $217 + 103 + 150 = 470$ )
6	7 (556), 8 (472), 5 (721), 3 ( $217 + 103 + 183 + 189 = 692$ ), Ziel ( $217 + 103 + 167 = 487$ )
8	7 (556), 5 (721), 3 (692), Ziel ( $85 + 80 + 250 + 84 = 499 > 487$ )
Ziel (487)	7 (556), 5 (721), 3 (692)

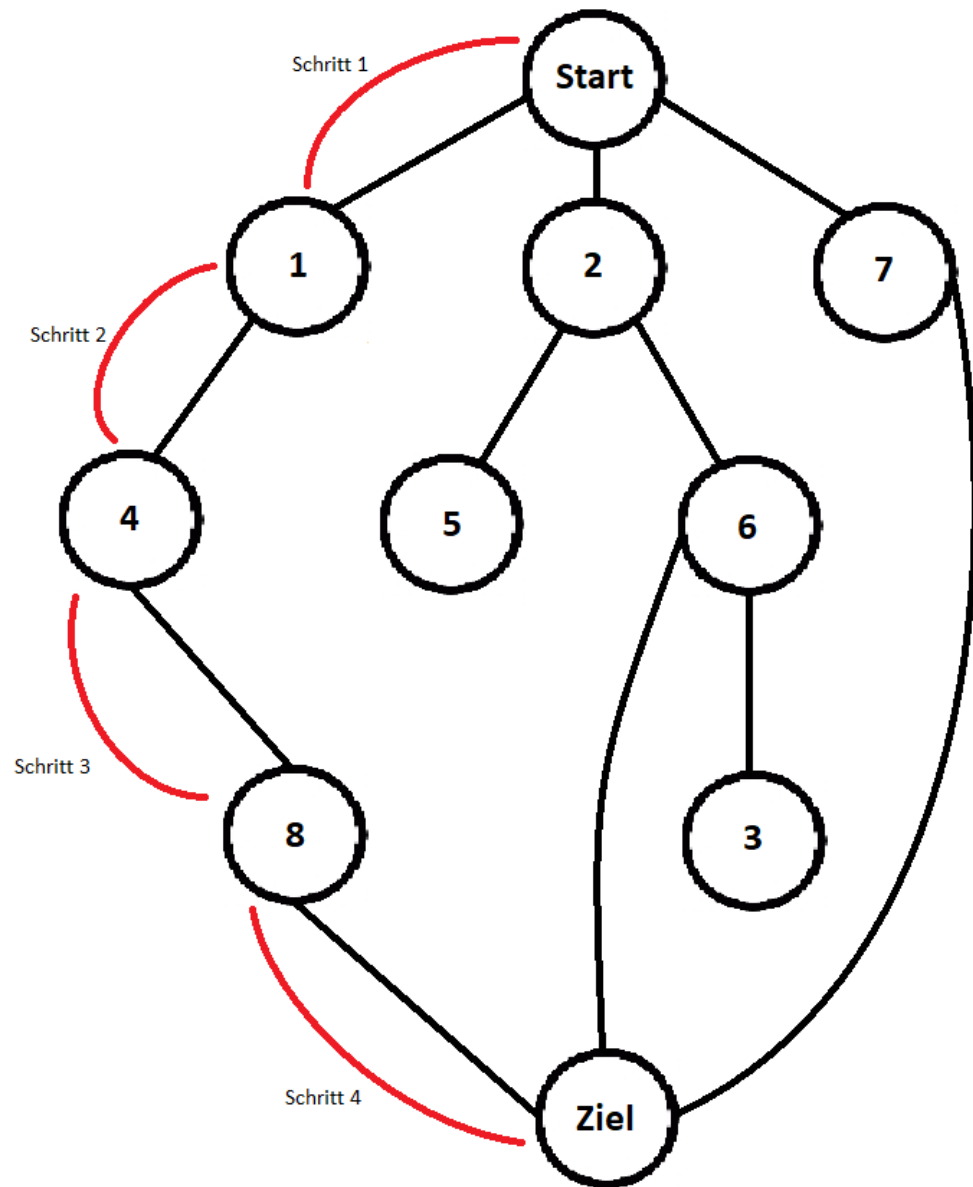
iv. Greedy: [Start, 2, 6, Ziel]

Node	Discovered
Start	1 (272), 2 (219), 7 (383)
2	1 (272), 7 (383), 5 (318), 6 (150)
6	1 (272), 7 (383), 5 (318), Ziel (0)
Ziel	1 (272), 7 (383), 5 (318)

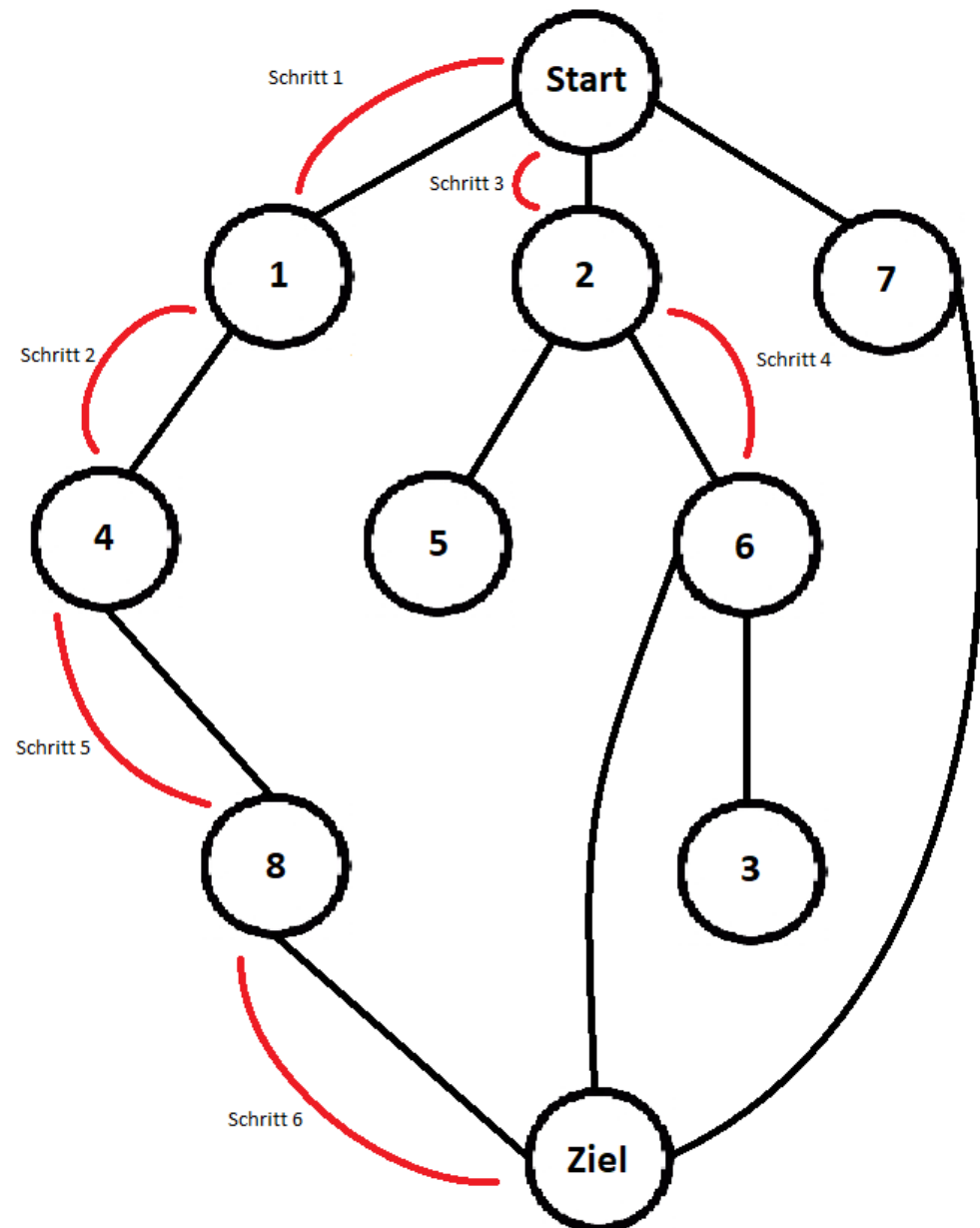
b) i. Breitensuche



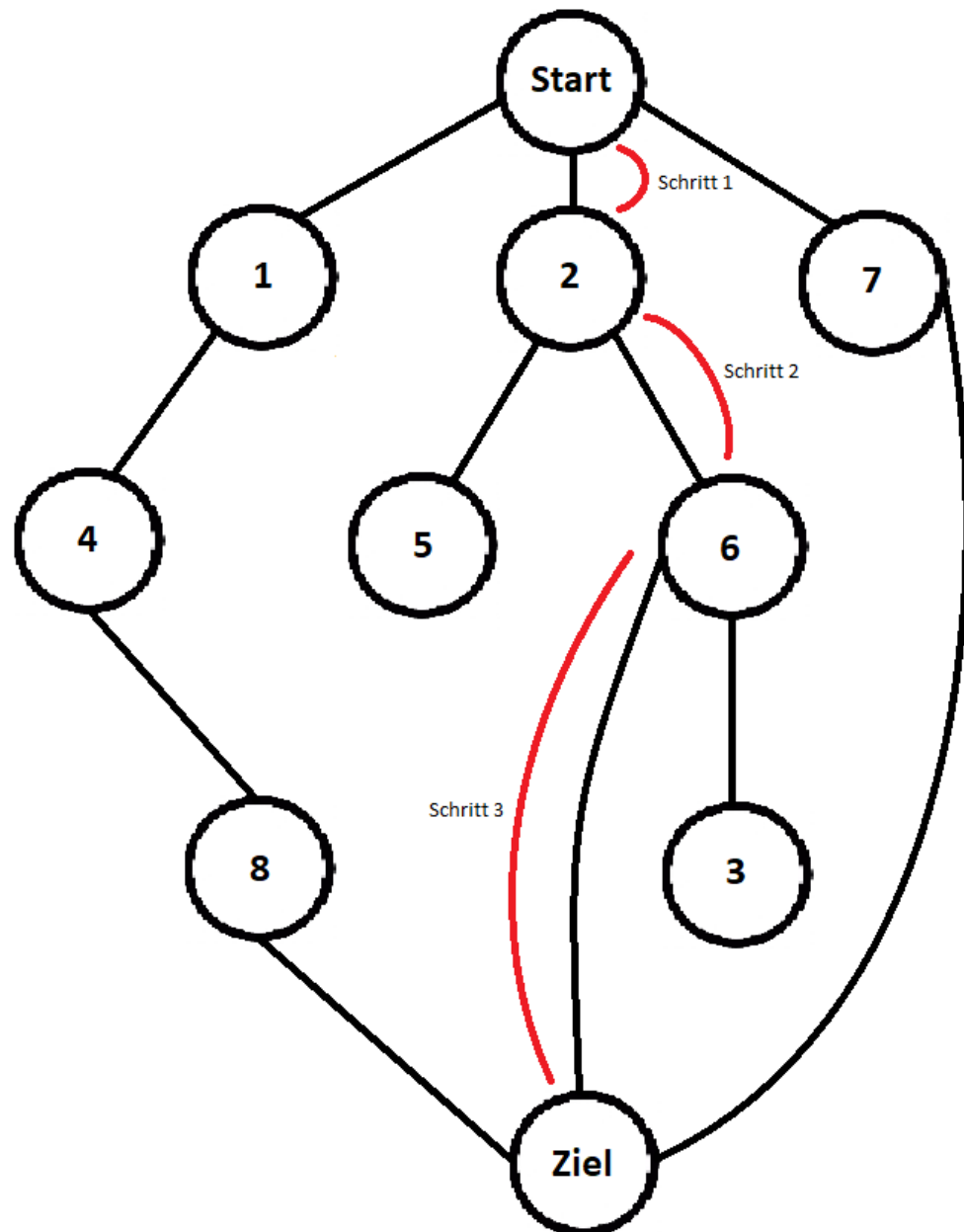
ii. Tiefensuche



iii. A\*



iv. Greedy



## 2.2 Implementierung

(a) Definition des Graphen

```

1 edges = {
2     'Start': {'1': 85, '2': 217, '7': 173},
3     '1': {'Start': 85, '4': 80},
4     '2': {'Start': 217, '5': 186, '6': 103},

```

```

5     '3': {'6': 183},
6     '4': {'1': 80, '8': 250},
7     '5': {'2': 186},
8     '6': {'2': 103, '3': 183, 'Ziel': 167},
9     '7': {'Start': 173, 'Ziel': 502},
10    '8': {'4': 250, 'Ziel': 84},
11    'Ziel': {'6': 167, '7': 502, '8': 84}
12 }
13
14 graph = LabelledGraph(edges)

```

---

#### (b) Heuristik

---

```

1 heuristics = {
2     'Start': 304,
3     '1': 272,
4     '2': 219,
5     '3': 189,
6     '4': 253,
7     '5': 318,
8     '6': 150,
9     '7': 383,
10    '8': 57,
11    'Ziel': 0
12 }
13
14
15 def heuristic(a, b):
16     if a == 'Ziel':
17         return heuristics[b]
18     raise NotImplementedError

```

---

#### (c) Greedy-Suche

---

```

1 def greedy_search(graph, start, goal):
2     frontier = PriorityQueue()
3     frontier.put(start, 0)
4     came_from = {}
5     cost_so_far = {}
6     came_from[start] = None
7     cost_so_far[start] = 0
8
9     while not frontier.empty():
10         current = frontier.get()

```



```

11     print("Visiting: %s with cost: %s" % (current, str(
12         cost_so_far[current])))
13     if current == goal:
14         print("Goal found: %s" % str(goal))
15         break
16
17     # calculate new cost for each neighbor
18     nn = graph.neighbors(current)
19     for nextkey in nn.keys():
20         nextcost = nn[nextkey]
21         new_cost = cost_so_far[current] + nextcost
22         if nextkey not in cost_so_far or new_cost <
23             cost_so_far[nextkey]:
24             cost_so_far[nextkey] = new_cost
25             # notice the change in the call to the heuristic
26             # function in the next line:
27             priority = heuristic(goal, nextkey)
28             frontier.put(nextkey, priority)
29             came_from[nextkey] = current
30
31     return came_from

```

---

(d) main-Funktion

---

```

1 def reconstruct_path(came_from, start, goal):
2     path = [goal]
3     while not path[0] == start:
4         path = [came_from[path[0]]] + path
5     return path
6
7
8 if __name__ == '__main__':
9     GREEDY = 'greedy'
10    A_STAR = 'a*'
11    algorithm = None
12    while not (algorithm in [GREEDY, A_STAR]):
13        algorithm = input('Give the search algorithm (%s / %s):
14            ' % (GREEDY, A_STAR))
15    if algorithm == GREEDY:
16        came_from = greedy_search(graph, 'Start', 'Ziel')
17    elif algorithm == A_STAR:
18        came_from, cost_so_far = a_star_search(graph, 'Start', '
19            Ziel')
20    print('Path: %s' % str(reconstruct_path(came_from, 'Start',
21        'Ziel')))

```

---