KI HA 1

Tom Schmidt Stefan Poggenberg Samuel Schöpa 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
           dirty."""
3      model = {}
4      direction = {(-1,0): 'Left', (1,0): 'Right', (0,-1): 'Up',
           (0,1): 'Down'}
5      last_location = None
6      last_move = None
7      moves = None
8
9      def program(percept):
10         nonlocal last_location
11         nonlocal last_move
12         nonlocal moves
13         (x, y), status = percept
14         model[(x, y)] = status
15
16         if not ((x, y) == last_location):
17             # new location: check for unknown neighbors
18             moves = [direction[(i, j)] for (i, j) in direction.
                   keys() if not (x + i, y + j) in model]
19             last_location = (x, y)
20         else:
21             # last move had no effect: remove last_move
22             if last_move in moves:
23                 moves.remove(last_move)
24
```

```
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

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(
                cost_so_far[current])))
12          if current == goal:
13              print("Goal found: %s" % str(goal))
14              break
15
16          # calculate new cost for each neighbor
```

```
17          nn = graph.neighbors(current)
18          for nextkey in nn.keys():
19              nextcost = nn[nextkey]
20              new_cost = cost_so_far[current] + nextcost
21              if nextkey not in cost_so_far or new_cost <
                    cost_so_far[nextkey]:
22                  cost_so_far[nextkey] = new_cost
23                  # notice the change in the call to the heuristic
                        function in the next line:
24                  priority = heuristic(goal, nextkey)
25                  frontier.put(nextkey, priority)
26                  came_from[nextkey] = current
27      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):
                ' % (GREEDY, A_STAR))
14      if algorithm == GREEDY:
15          came_from = greedy_search(graph, 'Start', 'Ziel')
16      elif algorithm == A_STAR:
17          came_from, cost_so_far = a_star_search(graph, 'Start', '
                Ziel')
18      print('Path: %s' % str(reconstruct_path(came_from, 'Start',
            'Ziel')))
```