# Aufgabe A3: Zauberschule

Team-ID: 00112

Team-Name: 10m Dürer gym

Bearbeiter/-innen dieser Aufgabe:
Finn Degen

19. November 2023

## Inhaltsverzeichnis



# 1 Lösungsidee

## 1.1 Allgemeines Vorgehen

Wir haben hier ein typisches Pathfinding Problem für welches sich der Algorithmus A* anbietet. Wir könnten natürlich auch DFS oder BFS verwenden, jedoch ist A* in der Regel schneller.
Abgesehen von den normalen Aktionen rechts,links,oben,unten, gibt es noch die Aktionen teleportieren welches die Sache etwas komplizierter macht.
Ein Node kann also maximal nicht 4 sondern 5 Nachbarn haben.
Für die Implementierung des A* Algorithmus sind in diesem Fall 2 Schritte notwendig. Wir brauchen unser Labirinth als Node Netzwerk und wir müssen eine passende Heuristik definieren.

## 1.2 Node Netzwerk

Für das Node Netzwek habe ich eine Klasse Node erstellt. Diese Klasse hat die folgende Attribute:

- *Truple: Position*: Speichert die Position des Nodes im Labyrinth x,y und Stockwerk
- *Node: Parent*: Speichert den Vorgänger Node

- *Int: Cost*: Speichert die Kosten für den Weg zum Node

Zwei Nodes sind gleich wenn sie die gleiche Position haben.
Zusätzlich zur Node Klasse gibt es die Funktion *get_ neighbours* welche die Nachbarn eines Nodes zurückgibt und somit das eigentliche Node Netzwerk bildet.

## 1.3 Heuristik

Wir benutzen eine recht einfache Heuristik. Die Heuristik ist die euklidische Distanz zwischen dem aktuellen Node und dem Ziel Node + 3 falls der Ziel Node auf dem anderen Stockwerk ist.
Formel: $h(n) = \sqrt{(x_{ziel} - x_{aktuell})^2 + (y_{ziel} - y_{aktuell})^2} + 3 * |stockwerk_{ziel} - stockwerk_{aktuell}|$

## 1.4 A* Algorithmus

Der A* Algorithmus ist ein informierter Suchalgorithmus. Er nutzt zwei Datenfelder:

- *heap: Queque*: Die Queque enthält alle Nodes die noch nicht besucht wurden. Die Nodes sind nach den Kosten sortiert.
- *set: Explored*: Das Explored Set enthält alle Nodes die bereits besucht wurden.

Der Algorithmus funktioniert kurzgefasst wie folgt:

1. Start Node in die Queque einfügen
2. Solange die Queque nicht leer ist:
   a) Node mit den geringsten Kosten aus der Queque entfernen
   b) Node in die Explored List einfügen
   c) Wenn der Node das Ziel ist: Algorithmus beenden
   d) Sonst: Nachbarn des Nodes in die Queque einfügen

# 2 Umsetzung

Der folgende Pseudocode beschreibt den Lösungsalgorithmus für dieses Problem.
Zuerst der A* Algorithmus:

```
queue ← []
explored ← {}
heapq.heappush(queue, start_node)
while queue is not empty do
    curNode ← heapq.heappop(queue)
    if curNode is equal to goal then
        GOAL_REACHED(curNode)
    end if
    explored.add(curNode)
    for neighbor in GET_NEIGHBORS(curNode) do
        if neighbor is in explored then
            continue
        end if
        if neighbor is not in queue then
            heapq.heappush(queue, neighbor)
        end if
    end for
end while
```

Die Funktion get_neighbors liest aus dem Text File vom Labyrinth die Nachbarn eines Nodes mithilfe dessen Position aus.

# 3 Beispiele

Im folgenden wird das Programm mit allen Beispielaufgaben ausgeführt

## 3.1 Beispiel 1

```
[[###############]
 [#.....#.....#]
 [#.###.#.###.#]
 [#...#.#...#.#]
 [###.#.###.#.#]
 [#...#.....#.#]
 [#.#########.#]
 [#.....#.....#]
 [#####.#.###.#]
 [#....!#B..#.#]
 [#.#########.#]
 [#...........#]
 [###############]]
[[###############]
 [#.......#...#]
 [#...#.#.#...#]
 [#...#.#.....#]
 [#.###.#.#.###]
 [#.....#.#...#]
 [#####.###...#]
 [#.....#.....#]
 [#.#########.#]
 [#...#>>!....#]
 [#.#.#.#.###.#]
 [#.#...#...#.#]
 [###############]]
```

## 3.2 Beispiel 2

```
[[######################]
 [#...#.....#...#.....#]
 [#.#.#.###.#.#.#.###.#]
 [#.#.#...#.#.#...#...#]
 [###.###.#.#.#####.###]
 [#.#.#...#.#B....#...#]
 [#.#.#.###.#^###.#####]
 [#.#...#.#.#^<<#.....#]
 [#.#####.#.#########.#]
 [#....................#]
 [######################]]
[[######################]
 [#.......#.....#.....#]
 [#.###.#.#.###.#.###.#]
 [#.....#.#.#.....#.#.#]
 [#######.#.#######.#.#]
 [#.....#.#.....#...#.#]
 [#.###.#.#.###.###.#.#]
 [#.#.#...#.#...#...#.#]
 [#.#.#######.###.###.#]
 [#...........#.......#]
 [######################]]
```

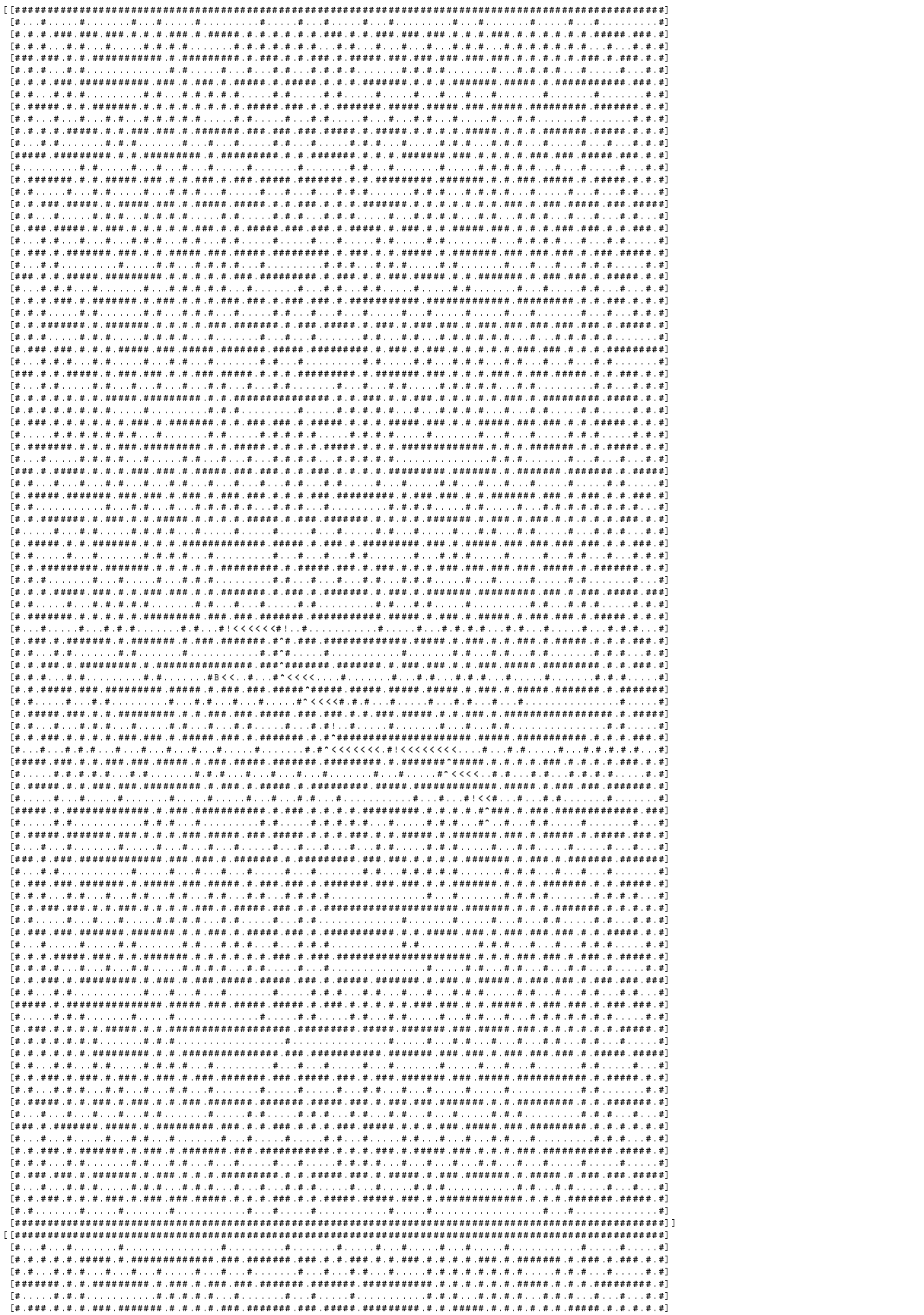
## 3.3 Beispiel 3

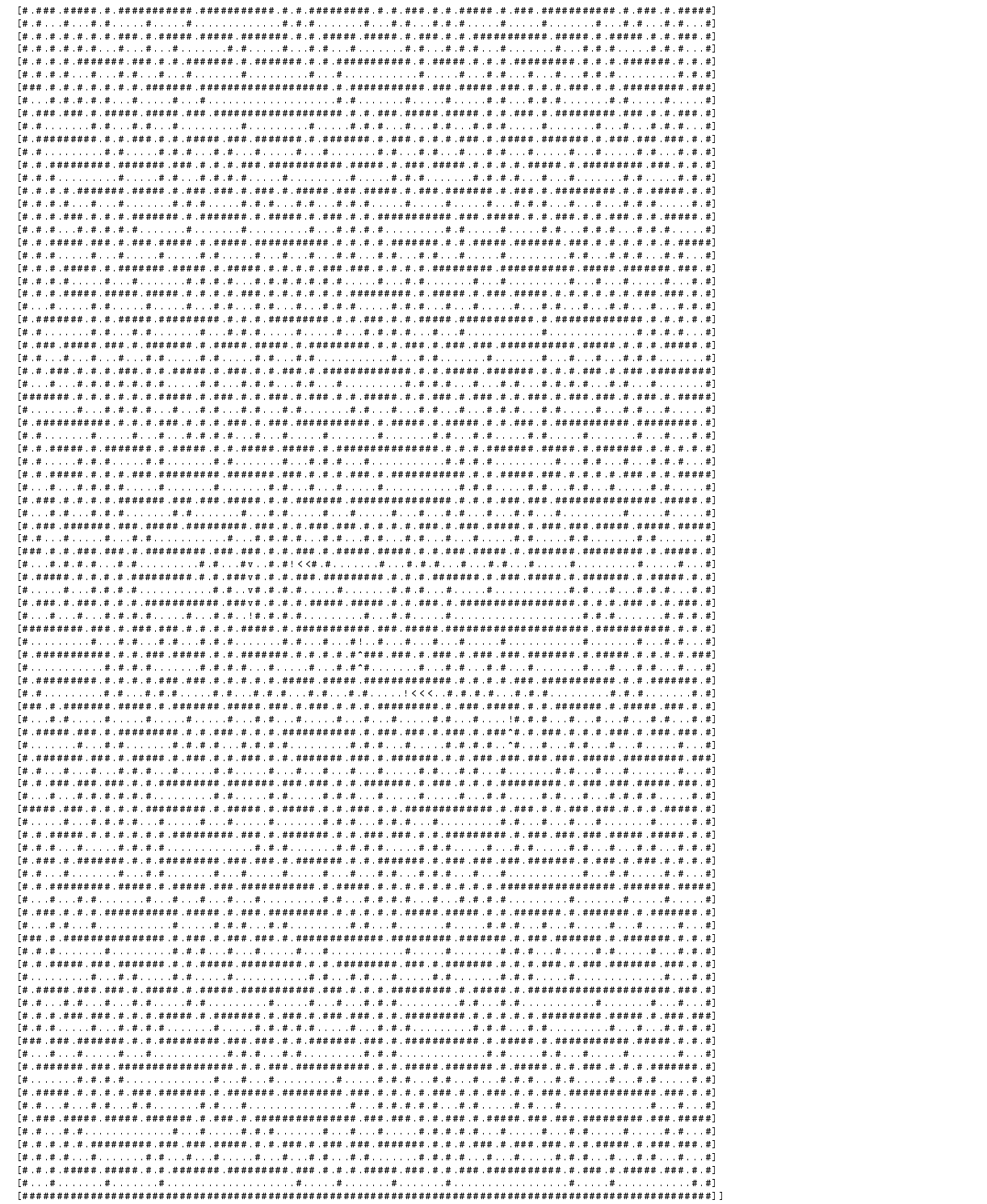
```
[[##############################################]
 [#...#.....#...........#.#...........#.......#]
 [#.#.#.###.#########.#.#.#.#######.#.#.#####.#]
 [#.#.#...#.#.........#>>!#v#.....#.#.#...#...#]
 [###.###.#.#.###############v#.#.###.#.###.#.###]
 [#.#.#...#.#..............>>B#.#...#.#...#.#.#]
 [#.#.#.###.###########.#########.###.#.###.#.#]
 [#.#...#.#.#.........#.#.#.....#.#.....#.#.#.#]
 [#.#####.#.#.#######.#.#.#.###.#.#######.#.#.#]
 [#.....#...#...#.#...#...#.#.#.#.#.......#.#.#]
 [#.#####.#####.#.#.#######.#.#.#######.#.#.#.#]
 [#.....#.......#.#.#.....#.#.#.#...#...#...#.#]
 [#.###.#########.#.###.#.#.#.#.#.#.#.###.###.#]
 [#...#.................#...#.....#...#.......#]
 [##############################################]]
[[##############################################]
 [#...#.......#.....#.....#...#...#.....#.....#]
 [#.#.#.#####.###.#.###.#.#.#.#.#.#.###.###.###]
 [#.#.#.....#.#...#.....#>>!#.#.#...#.#...#...#]
 [###.#.###.#.#.###############.#.#####.###.###.#]
 [#.#.#...#.#.#.#.....#...#.#.#...#.#...#.#...#]
 [#.#.#####.#.#.#.###.#.#.#.#.#.#.#.#.#.#.#.###]
 [#.#...#...#.#.....#.#.#...#.#.#.#...#.#.#...#]
 [#.###.#.###.#.#####.#.###.#.###.#####.#.###.#]
 [#...#.#.#...#...#...#...#.#...#.#.....#.....#]
 [#.###.#.#.#######.#####.#####.#.#.#.#######.#]
 [#...#...#...#...#.....#.......#.#.#.#.....#.#]
 [#.#.#######.#.#.#####.#.#######.#.###.###.#.#]
 [#.#...........#.......#.........#.......#...#]
 [##############################################]]
```

## 3.4 Beispiel 4

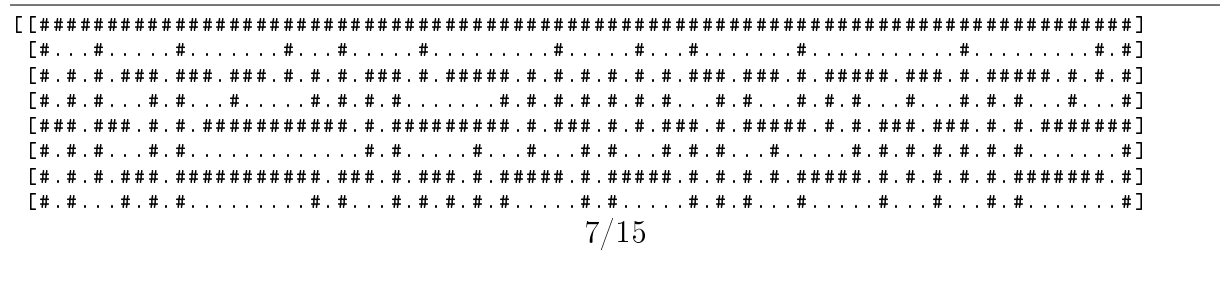
```
[[##################################]
 [#...#.....#...........#.......#]
 [#.#.#.###.#.###.#######.###.#.#]
 [#.#.#...#.#.#.#.#.......#...#.#]
 [###.###.#.#.#.#.#.#######.###.#]
 [#.#.#...#.#...#.....#.....#.#.#]
 [#.#.#.###.############.#####.#.#]
 [#.#...#.#.#.........#...#.....#]
 [#.#####.#.#.#######.###.#.#####]
 [#...#.#...#...#.#...#...#.#...#]
 [#.#.#.#.#####.#.#.###.###.#.#.#]
 [#.#.#.#.......#.#.#...#.#...#.#]
 [#.#.#.##########.#.#.###.#####.#]
 [#.#.......#.#.....#.#.#.....#.#]
 [#.#######.#.#.#####.#.#.###.#.#]
 [#.#.....#...#.#.#...#.#.#.#...#]
 [#.#.###.#####.#.#.#.#.#.#.#####]
 [#.#...#.....#.#.#.#.#...#.....#]
 [#.###.#####.#.#.#.#.###.#####.#]
 [#...#.#...#...#...#...#.......#]
 [#.#.#.#.##############.#.#######]
 [#.#.#.#.....................#.#.....#]
 [#.###.#.#######.##########.###.#]
 [#...#.#.#...#...#.........#...#]
 [###.#.###.#.#.###.#####.###.#.#]
 [#...#.....#.#.....#>>B#.#...#.#]
 [#.#########.#######^#.###.###.#]
 [#..!#>>>>v#.#>>>>>>^#...#.#.#.#]
 [#.#.#.###v#.#^##########.#.#.#.#]
 [#.#.....#>>>>^#...........#...#]
 [##################################]]
[[##################################]
 [#.......#.....#...#...#.......#]
 [#.###.#.#.#.#.#.###.#.#.#.#####]
 [#.....#.#.#.#.#.....#.#.#.....#]
 [########.#.#.#.#.#####.#.#####.#]
 [#.....#.#.#.#.#.#.#...#...#.....#]
 [#.###.#.###.#.###.#.#.###.#####]
 [#...#.#.#...#.....#.#.#.#.....#]
 [#.#.###.#.#########.#.#.#####.#]
 [#.#.....#.#.......#.#.....#...#]
 [#.#######.#######.#.#.#####.#.#]
 [#...#...#.....#...#.#.#...#.#.#]
 [###.#.#.#.###.#.###.#.#.#.#.#.#]
 [#.#.#.#...#...#.....#.#.#.#.#.#]
 [#.#.#.#####.###.#######.#.#.#.#]
 [#.#.#.....#.#.....#.....#.#.#.#]
 [#.#.#######.#####.###.###.#.#.#]
 [#.#.....#...#...#.....#.#...#.#]
 [#.#####.#.###.#.#######.#####.#]
 [#.#...#.#.#...#.......#.#.....#]
 [#.#.#.#.#.#.#########.#.#.#####]
 [#...#.#.#.#...#...#.#.#.#.....#]
 [#####.#.#.###.#.#.#.#.#.#####.#]
 [#.....#.#.......#.#.#.#...#...#]
 [#.###.#.#########.#.#.#.#.#.###]
 [#...#.#.....#.#.....#.#.#.#.#.#]
 [#.#.#######.#.#.#####.#.#.#.#.#]
 [#.#>>!....#.#...#.....#.#...#.#]
 [#.###.#####.#####.#####.#####.#]
 [#...#.............#...........#]
 [##################################]]
```

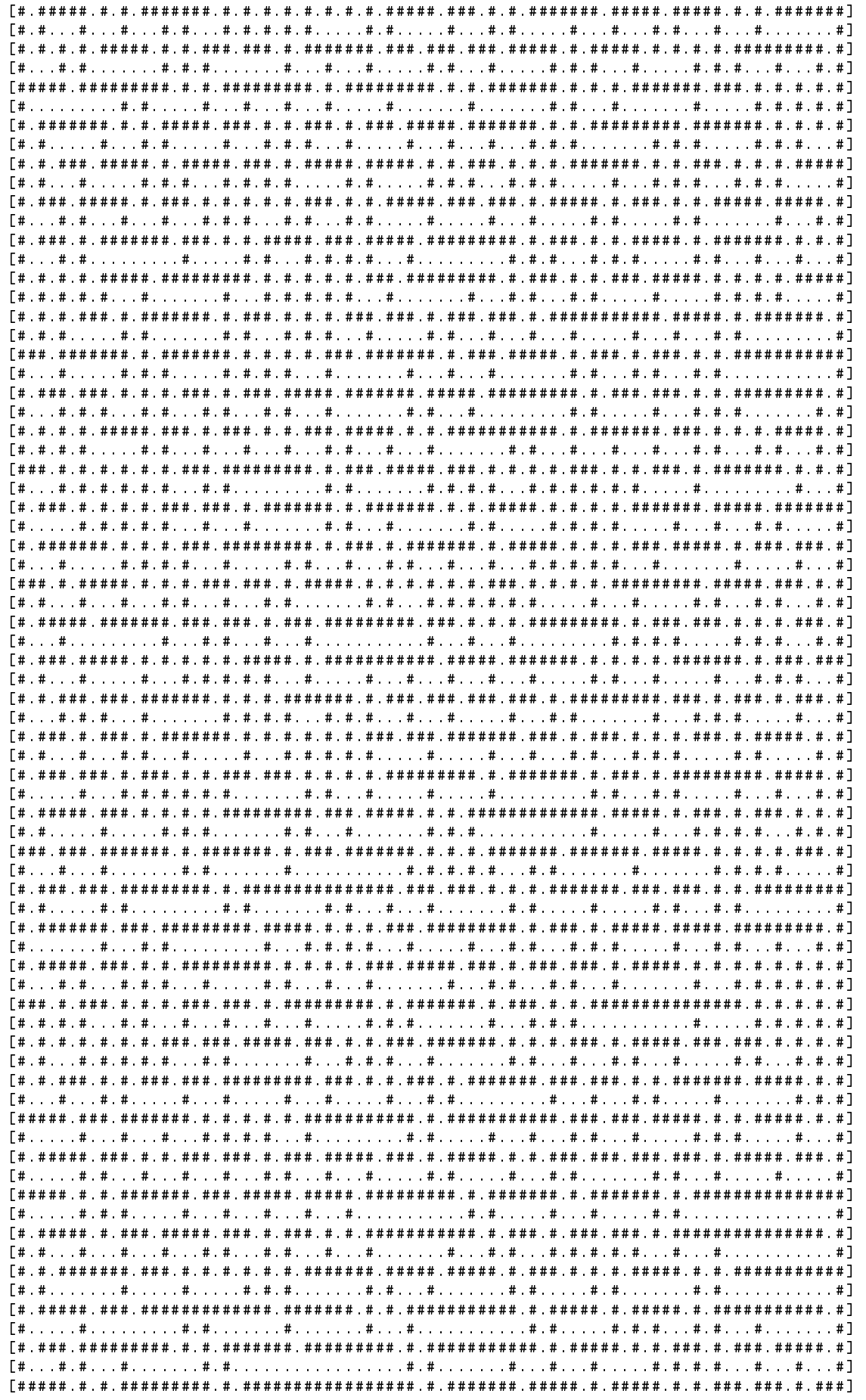
## 3.5 Beispiel 5

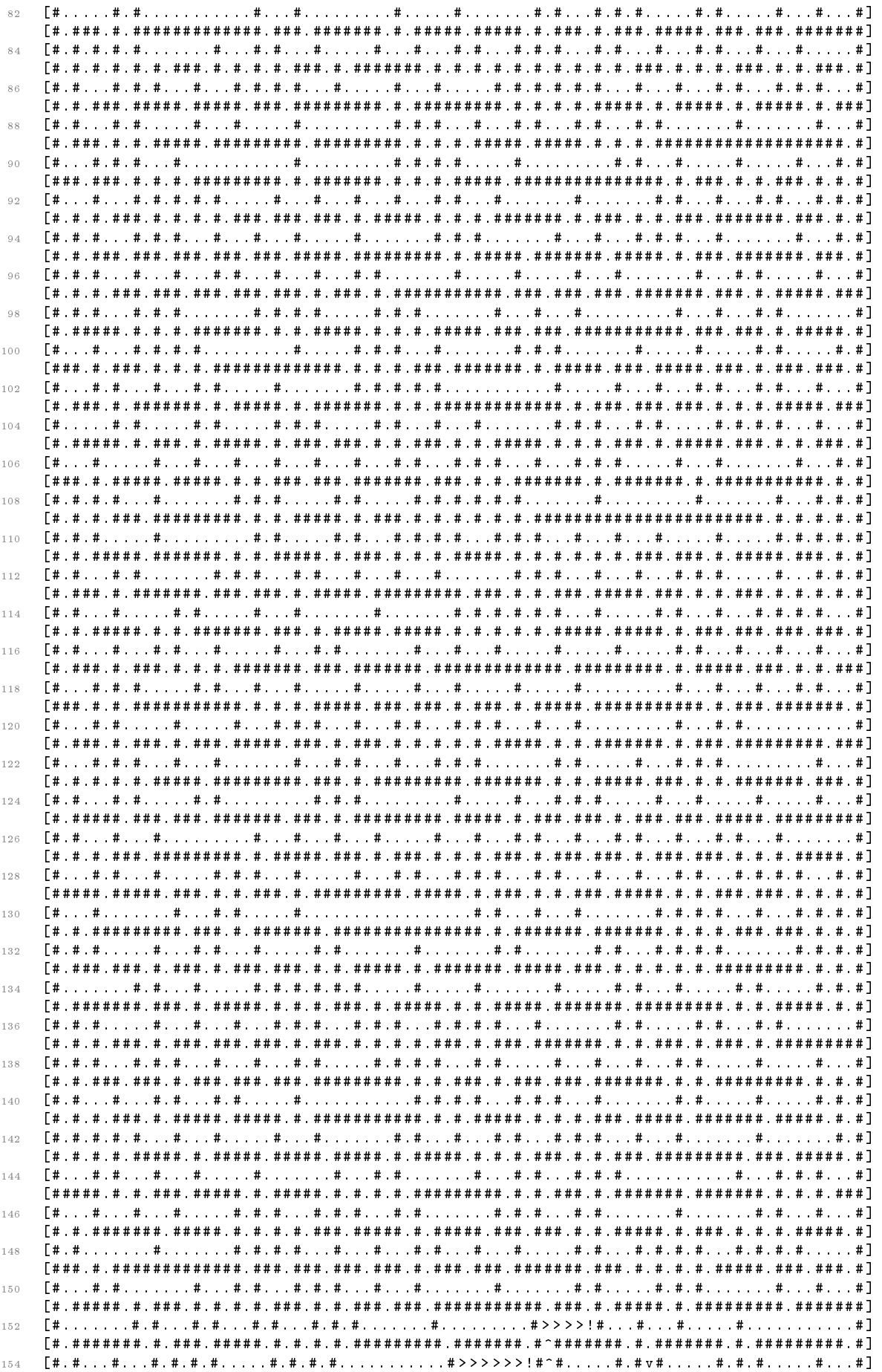


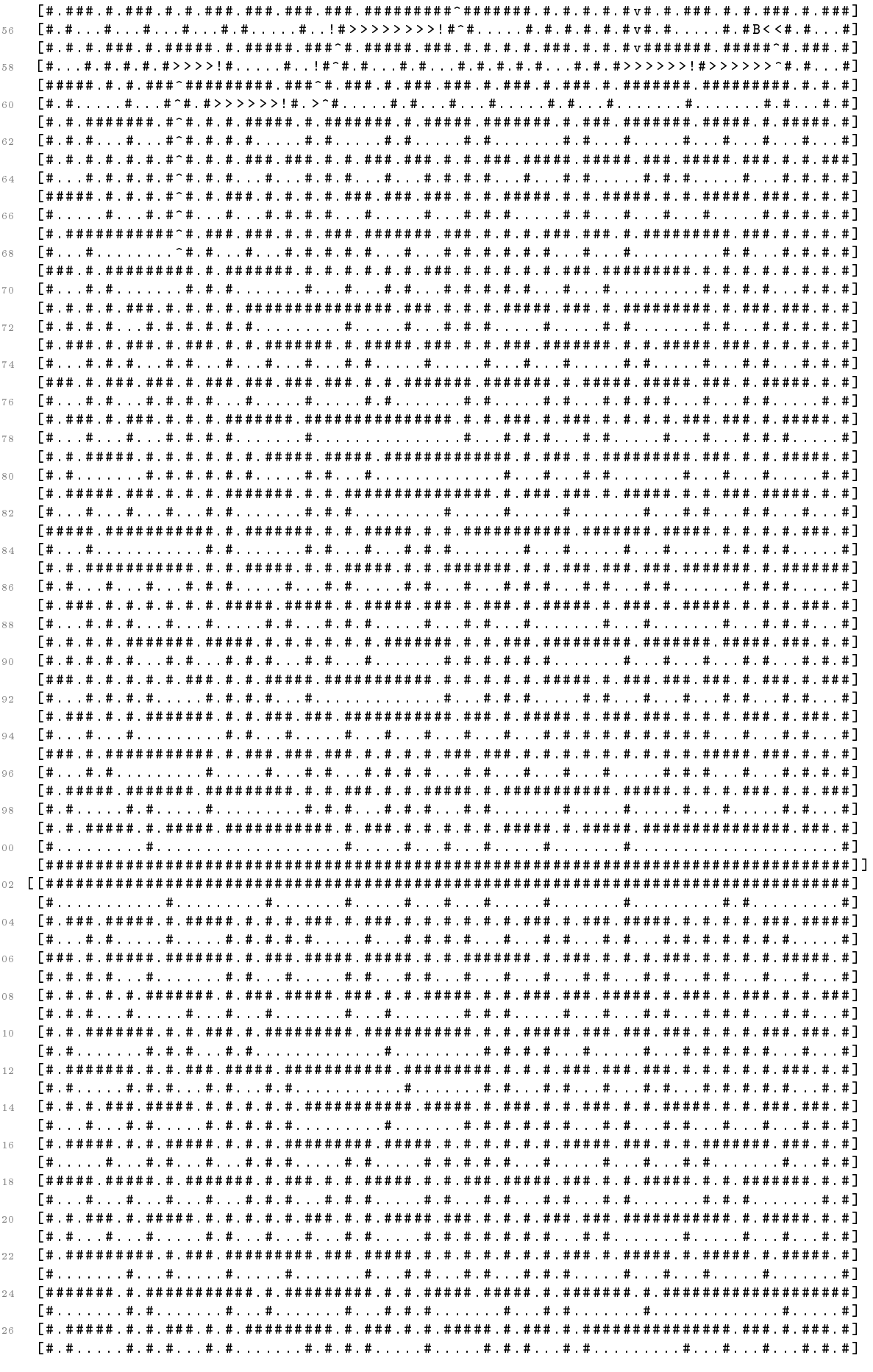


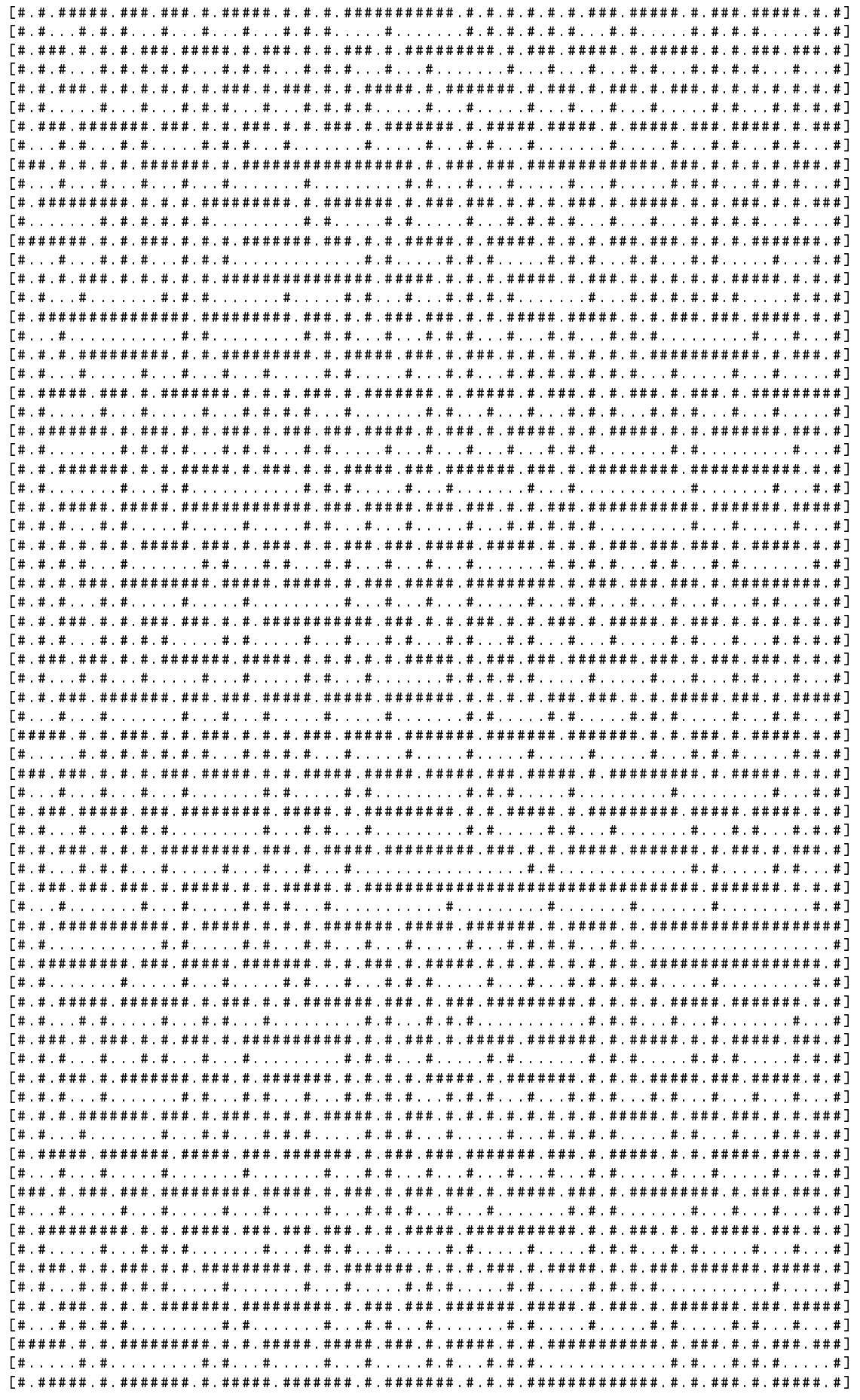
## 3.6 Beispiel 6

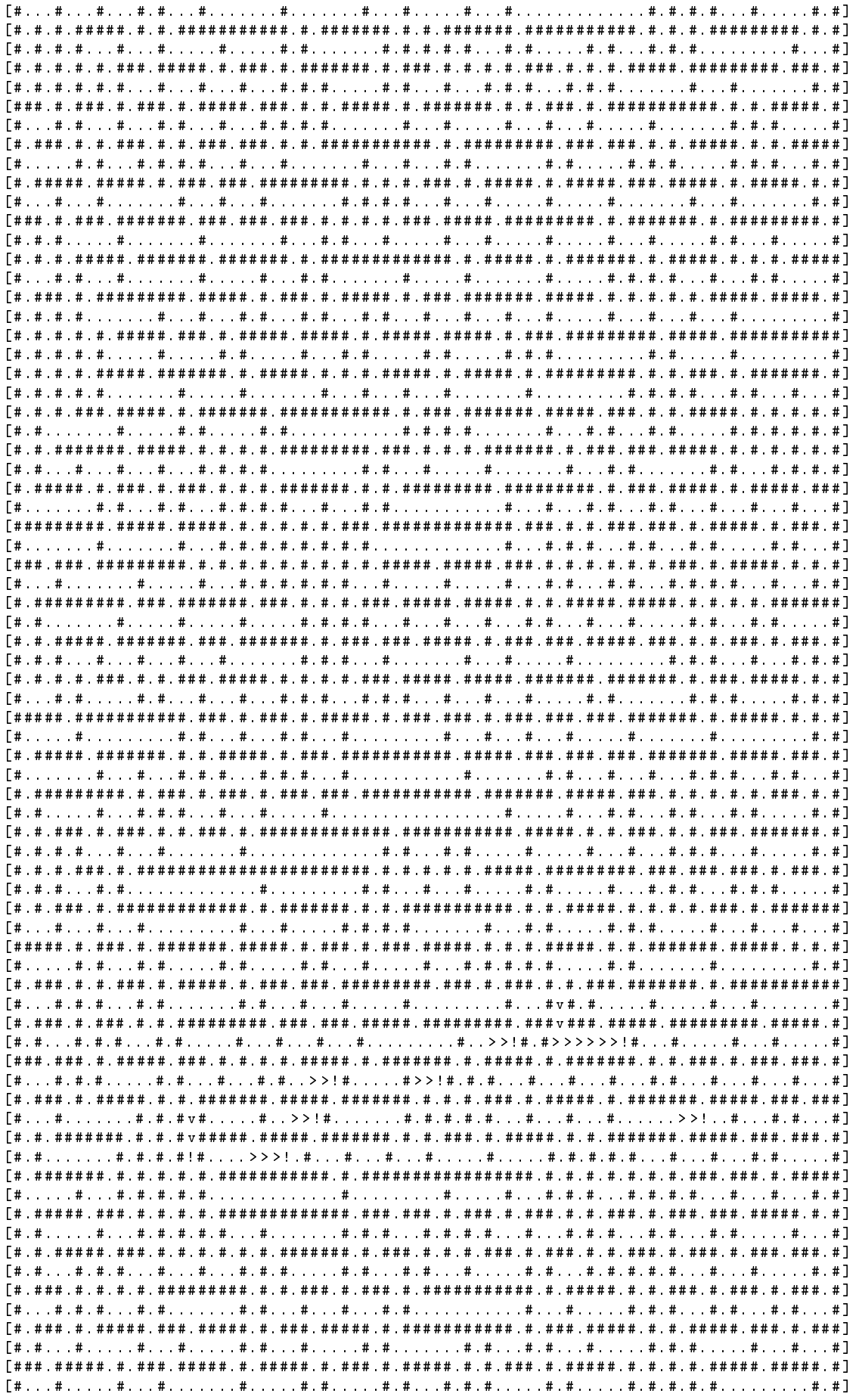












```
[#.###.###########.#########.#.#######.#.#.#.#.#######.#.#.#####.#.#.###########.#]
[#.#...............#.......#.#.........#.#.#.#...#.....#.#.....#...#.............#]
[#.#################.#####.#.#.#########.#.#.#.###.#######.###.#####.#############]
[#.........#...#.......#...#.#.#...#.....#...#.#...#.....#.#.#...#...#.....#.....#]
[#.#######.#.#.#.#######.#.#.###.#.#.#########.#.###.###.#.#.###.#.###.###.#####.#]
[#.#...#...#.#...#...#...#.#.....#.#.#.........#.....#...#.....#.#...#.#.......#.#]
[#.###.#.#.#.#####.###.###.#######.#.#####.###########.#######.#.###.#.#######.#.#]
[#.#...#.#.#.#.....#...#...#.#.....#.#.....#...#.....#...#.....#.#...#.#.....#.#.#]
[#.#.#.#.#.#.###.###.###.###.#.#####.#.###.#.#.#.###.###.#.#####.#.###.#.###.#.#.#]
[#.#.#.#.#.#.#...#...#.......#...#.#.#...#.#.#.....#...#.#.#.#...#.#.....#...#.#.#]
[#.#.###.###.#.###.###.#####.###.#.#.###.###.#######.###.#.#.#.#.#.#######.###.#.#]
[#.#...#...#.#.#.#...#...#...#.#.#.#.....#.......#...#...#.#...#.#.#.........#.#...#]
[#.#.#.###.#.#.#.#######.#.#.#.#.#####.#####.###.#.#.#.#.###.#.#.###########.###.#]
[#.#.#.#...#...#.........#.#.#.....#.#...#.#...#.#.#.#...#.#.#.#...........#...#]
[#.###.#.#####.###########.#.#.#####.#.#.#.###.###.###.#.#.###.###.#######.###.###]
[#.....#.#...#...#.....#...#.......#.#.#.#...#...#.....#.#.#.#...........#.......#]
[###.###.#.#.###.#.#.###.###########.#.#.###.###.#######.#.#.######################]
[#...#...#.#.......#.#...#.....#...#.#.#.#.#.....#.....#.#.......#...............#]
[#.###.###.###########.###.###.#.#.#.#.#.#.#####.#.###.#.#########.##############.#]
[#.#.#.#...#.#.........#...#.#...#.#.#.#.#.......#.#...#...........#.....#.....#.#]
[#.#.#.#.###.#.#########.###.#####.#.#.#.#.#######.#.###.###########.###.#.###.#.#]
[#...#.#...#.#...........#.....#...#.#.#.#.....#...#.#.#.#.....#.....#.#...#.#.#.#]
[###.#.###.#.###########.#.#####.###.#.#.#######.###.#.#.#.###.#.#####.#####.#.#.#]
[#...#...#...#.....#...#...#...#.#...#...#...#...#.#.#...#...#.#.....#...#...#.#]
[#.###.#.###.###.###.#.#####.#.#.#.#####.#.#.#.###.#.#.#####.#######.#.#.#.#.###.#]
[#.#...#...#...#.#...#.......#...#.#.....#.#.#.#...#.#...#...#.#.....#.#.#...#...#]
[#.#.#########.#.#.################.#######.#.#.#.#.#.#####.###.#.#.#####.#######.#]
[#.#............#.........................#...#.#.........#.....#.............#]
[#############################################################################]]
```

## 4 Quellcode

Zuerst folgt die Implementation der Klasse Node so wie in Abschnitt 1.2 beschrieben.

```python
class Node:
"""Eine einfache Node Klasse für backtracking und Cost berechnen, speichert auch die Position

:var position: Die Position des Nodes
:var parent: Der Parent Node
:var cost: Die Kosten um zu diesem Node zu kommen
"""

def __init__(self, position, parent=None, cost=0):
    self.position = position  # y,x,floor
    self.parent = parent  # parent node
    self.cost = cost  # the basic cost + heuristicCost of this Node

def __repr__(self) -> str:
    return f"Node: x: {self.position[0]} y: {self.position[1]} floor: {self.position[2]} cost: {self.co

# all methods for sorting in the priority queue and comparing nodes
def __eq__(self, __value: object) -> bool:
    return self.position == __value.position

def __lt__(self, __value: object) -> bool:
    return self.cost < __value.cost

def __le__(self, __value: object) -> bool:
    return self.cost <= __value.cost

def __gt__(self, __value: object) -> bool:
    return self.cost > __value.cost

def __ge__(self, __value: object) -> bool:
    return self.cost >= __value.cost

def __hash__(self) -> int:
    return hash(self.position)
```

Nun die Heuristik Methode welche genau so wie in Abschnitt 1.3 beschrieben implementiert wurde.

```
def heuristic_cost(nodePos: tuple[int, int, int], goalPos: tuple[int, int, int]) -> int:
"""Berechnet die Heuristik für die Distanz zwischen zwei Punkten

:param nodePos: Die Position des aktuellen Nodes
:param goalPos: Die Position des Zielnodes
:returns: Die estimierten Kosten um von nodePos zu goalPos zu kommen
"""
x1, y1, f1 = nodePos
x2, y2, f2 = goalPos

return abs(x1 - x2) + abs(y1 - y2) + abs(f1 - f2) * 3  # floor change cost is 3
```

Als Nächstes die get_neighbours Methode welche ja bisher nur kurz erwähnt wurde. Sie gibt alle möglichen Nachbarn eines Nodes zurück (möglich wenn das Feld keine Wand ist und es nicht auserhalb des Spielfelds ist). Außerdem speichert die Funktion die Kosten um zu diesem Node zu kommen direkt in dem erstellten Node ab. Auch der Parent Node wird direkt gesetzt. *floors* ist die Karte der Schule als 3D Array.

```
def get_neighbors(node, floors, goalPos: tuple[int, int, int]):
"""Gibt alle Nachbarn eines Nodes zurück

:param node: Der Node dessen Nachbarn gesucht werden
:param floors: Die Etagen der Schule als Karte
:param goalPos: Die Position des Zielnodes
:returns: Eine Liste aller Nachbarn
"""
x, y, f = node.position
neighbors = []

for ajacent_x, ajacent_y in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
    new_x, new_y = x + ajacent_x, y + ajacent_y
    if floors[f][new_x][new_y] != "#":
        neighbors.append(
            Node(
                (new_x, new_y, f),
                parent=node,
                cost=node.cost + 1 + heuristic_cost((new_x, new_y, f), goalPos),
            )
        )

if floors[1 - f][x][y] != "#":
    neighbors.append(
        Node(
            (x, y, 1 - f),
            parent=node,
            cost=node.cost + 3 + heuristic_cost((x, y, 1 - f), goalPos),
        )
    )  # for two floor 1-f because 0->1 and 1->0

return neighbors
```

Nun folgt die Implementation des A* Algorithmus welche nur der Pseudocode aus Abschnitt 1.4 in Python ist.

```
def a_star(start: Node, goal: Node, floors) -> list[tuple[int, int, int]]:
"""Führt den A* Algorithmus aus

:param start: Der Startnode
:param goal: Der Zielnode
:param floors: Die Etagen der Schule als Karte
:returns: Der Pfad von start zu goal
"""
queque = []
explored = set()

heapq.heappush(queque, start)

while queque:
    curNode: Node = heapq.heappop(queque)
```

```
16
    if curNode == goal:
18      # Goal reached, construct and return the path
        path: list[tuple[int, int, int]] = []
20      while curNode:
            # backtrack back to start
22          path.append(curNode.position)
            curNode = curNode.parent
24      return path[::-1]  # reverse because we backtracked from end

26  # we have now seen this node
    explored.add(curNode)
28
    for neighbor in get_neighbors(curNode, floors, goal.position):
30      if neighbor in explored:
            # already seen dont care
32          continue

34      if neighbor not in queque:
            heapq.heappush(queque, neighbor)
```

Zuletzt folgt noch die Implementation der main Methode welche die Karte aus dem txt ausliest und dann den A* Algorithmus ausführt.

```
1 # InputProcessing
    x, y = [int(data) for data in rawData[0].split()]
3
    floors = [
5       np.array([list(line) for line in rawData[1 : x + 1]]),
        np.array([list(line) for line in rawData[x + 2 : 2 * x + 2]]),
7   ]
    startPos = np.argwhere(floors[0] == "A")[0]
9   endPos = np.argwhere(floors[0] == "B")[0]

11  startNode: Node = Node((startPos[0], startPos[1], 0))
    endNode: Node = Node((endPos[0], endPos[1], 0))
13
    # run aStar
15  path: list[tuple[int, int, int]] = a_star(startNode, endNode, floors)
```

Danach muss nur noch der Pfad schön ausgegeben werden und die Aufgabe ist gelöst.