2025-2\LectureScripts\Script3_HeuristicSearch\code\agents.py

```python
import numpy as np
from heapq import heappush, heappop
from itertools import count

class Node:

    def __init__(self, state, parent=None, g=0.0, h=0.0):
        self.state  = state
        self.parent = parent
        self.g      = g
        self.h      = h
        self.depth  = 0 if parent is None else parent.depth + 1

    def __eq__(self, other):
        if not isinstance(other, Node):
            return NotImplemented
        return np.array_equal(self.state, other.state)

    def __lt__(self, other):
        return True

    def __hash__(self):
        return hash(self.state.tobytes())

def get_solution(node):
    solution = []
    solution.append(list(node.state))
    while node.parent:
        node = node.parent
        solution.append(list(node.state))
    solution.reverse()
    return solution


def remove_last(F):
    return F.pop(-1)

def add_last(F,s):
    F.append(s)

def remove_first(F):
    return F.pop(0)

def cost_manhattan(parent_node):
    return parent_node.g + 1

def h_manhattan(state,exit):
    return np.sum(np.abs(state - exit))

def add_heap_astar(F,s):
    heappush(F, (s.h+s.g, s))
```

```python
def add_heap_greedy(F,s):
    heappush(F, (s.h, s))

def remove_heap(F):
    return heappop(F)[1]

class AgentMaze:

    def __init__(self, env, add_fcn, remove_fcn, cost_fcn, h_fcn):
        self.env = env
        self.visited = set()
        self.initial_percepts = env.initial_percepts()
        self.G = self.initial_percepts['exit']
        self.remove_fcn = remove_fcn
        self.add_fcn = add_fcn
        self.h_fcn = h_fcn
        self.cost_fcn = cost_fcn


    def search(self):

        s0 = Node(self.initial_percepts['start'], g=self.cost_fcn(Node(self.initial_percepts['exit'])), h=self.h_fcn(self.initial_percepts['exit'],self.G))

        F = []

        self.add_fcn(F,s0)

        while F:

            s = self.remove_fcn(F)
            print(s.state)

            if (s.state == self.G).all():
                return s

            self.visited.add(s)

            for s_ in self.env.get_neighbors(s.state):
                s_node = Node(s_,s,self.cost_fcn(s),self.h_fcn(s_,self.G))
                if s_node not in self.visited:
                    self.add_fcn(F,s_node)

        return None
```