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

```python
import numpy as np
import matplotlib.pyplot as plt

class Maze:

    def __init__(self, matrix, start, exit):
        self.matrix = matrix
        self.start = start
        self.exit = exit

    def initial_percepts(self):

        return {'start': np.array(self.start),
                'exit': np.array(self.exit)}

    def get_neighbors(self, state):

        actions = np.array([[-1,0],[1,0],[0,-1],[0,1]])
        neighbors = []
        for action in actions:
            neighbor = np.array(state) + action
            if (0 <= neighbor[0] < len(self.matrix)) and (0 <= neighbor[1] < len(self.matrix[0])) and (self.matrix[neighbor[0]][neighbor[1]] == 0):
                neighbors.append(neighbor)

        return neighbors

    def plot(self, path=None, ax=None, show=True):
        """
        Plot the maze and (optionally) a path as a sequence of (row, col) positions.
        - path: list/array of positions [[r0,c0], [r1,c1], ...]
        - ax:   optional matplotlib Axes to draw on
        - show: call plt.show() if True
        """
        m = np.array(self.matrix)
        if ax is None:
            fig, ax = plt.subplots(figsize=(5, 5))

        # walls (1) -> black; free (0) -> white
        ax.imshow(m == 1, cmap="gray", origin="upper")

        # draw path if provided
        if path is not None and len(path) > 0:
            path = np.array(path)
            # NOTE: plot expects x=col, y=row
            ax.plot(path[:, 1], path[:, 0], linewidth=2)

        # mark start and exit
        ax.scatter(self.start[1], self.start[0], marker="o", s=80, label="start")
        ax.scatter(self.exit[1],  self.exit[0],  marker="*", s=140, label="exit")

        # grid lines
```

```
        ax.set_xticks(np.arange(-.5, m.shape[1], 1), minor=True)
        ax.set_yticks(np.arange(-.5, m.shape[0], 1), minor=True)
        ax.grid(which="minor", linewidth=0.5)
        ax.set_xticks([]); ax.set_yticks([])
        ax.legend(loc="upper right")
        plt.tight_layout()
        if show:
            plt.show()
        return ax
```