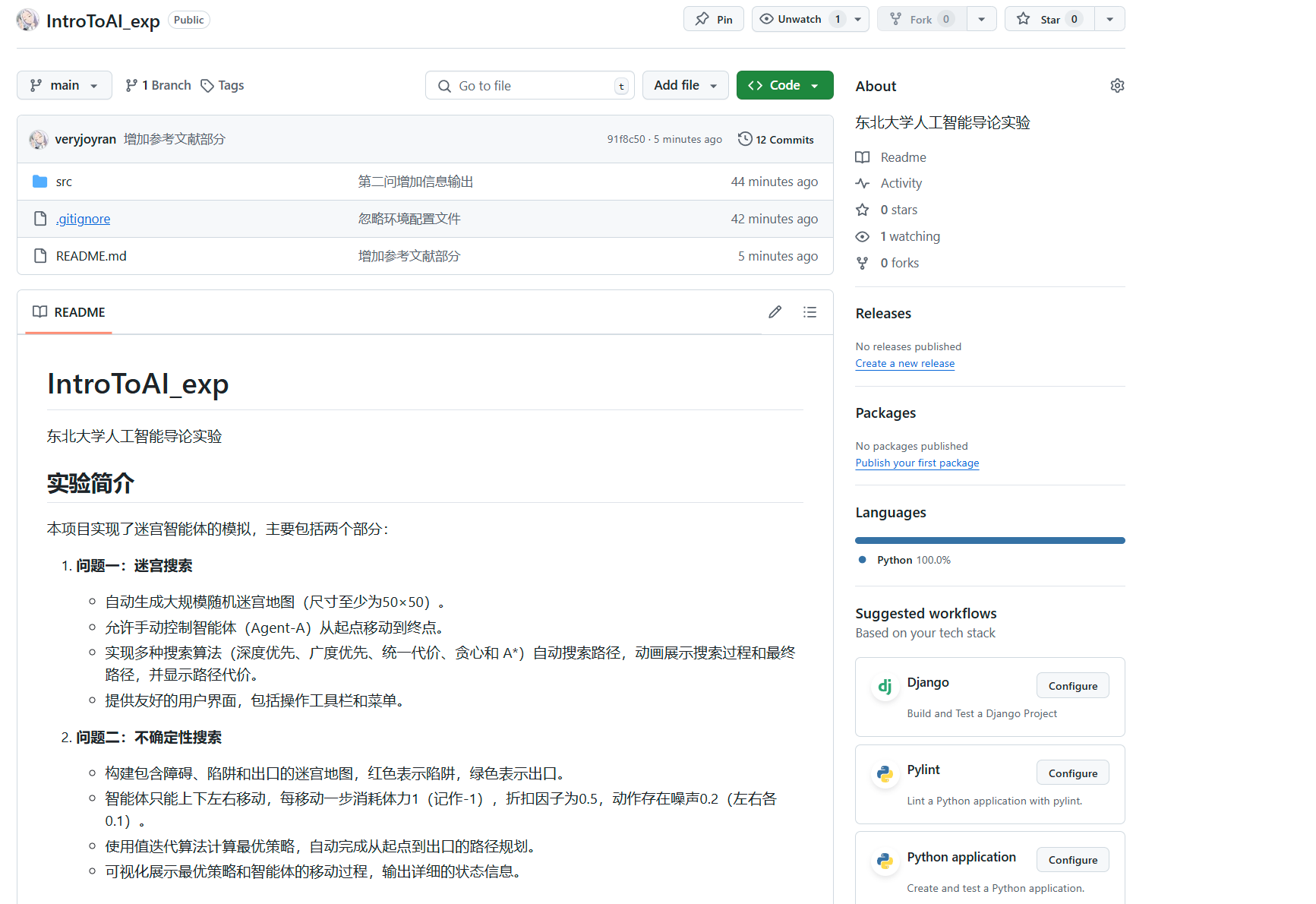
实验二：不确定性搜索

人工智能2202班-刘庆然-20226494

所有代码以及README均已上传至github: <https://github.com/veryjoyran/IntroToAI_exp>



目录

[一、 问题描述 3](#_Toc181927064)

[二、问题描述与解决方案 4](#_Toc181927065)

[三、实验结果 5](#_Toc181927066)

[四、源码解读 9](#_Toc181927067)

[Problem2.py: 9](#_Toc181927068)

[源代码： 11](#_Toc181927069)

# 问题描述

编写程序实现一个迷宫智能体。 具体要求如下：

1. 构建如右的迷宫地图（可自行设计， 元素包括障碍强、 陷阱和出口）， 其中红

色为陷阱， 绿色是出口。

2. 智能体只能上下左右四个方

向行为， 每行动一步消耗掉

体力 1， 记作-1， 折扣因子记

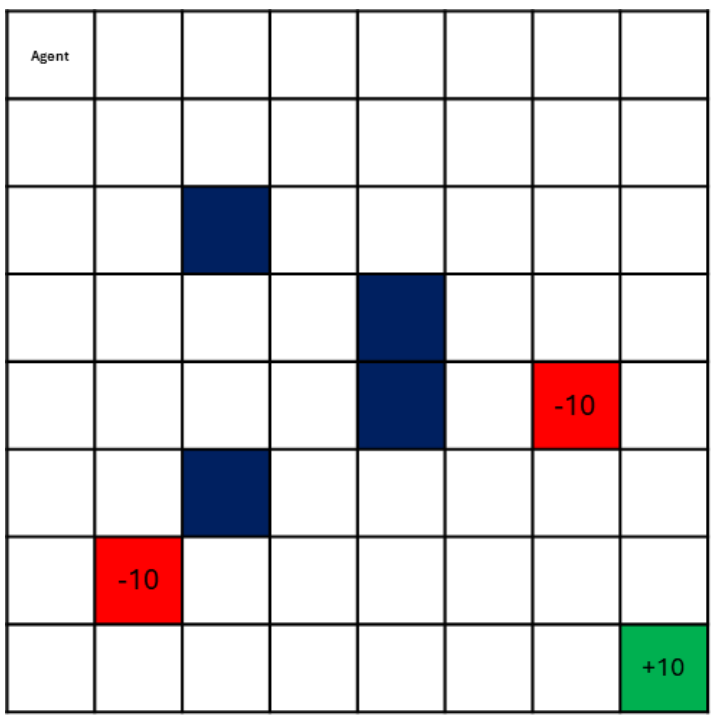
为 0.5， 噪声 0.2（左右各

0.1）， 找到出口结束。

3. 设计算法自动完成上述要求

过程， 给出最优的行为策略

及路线。

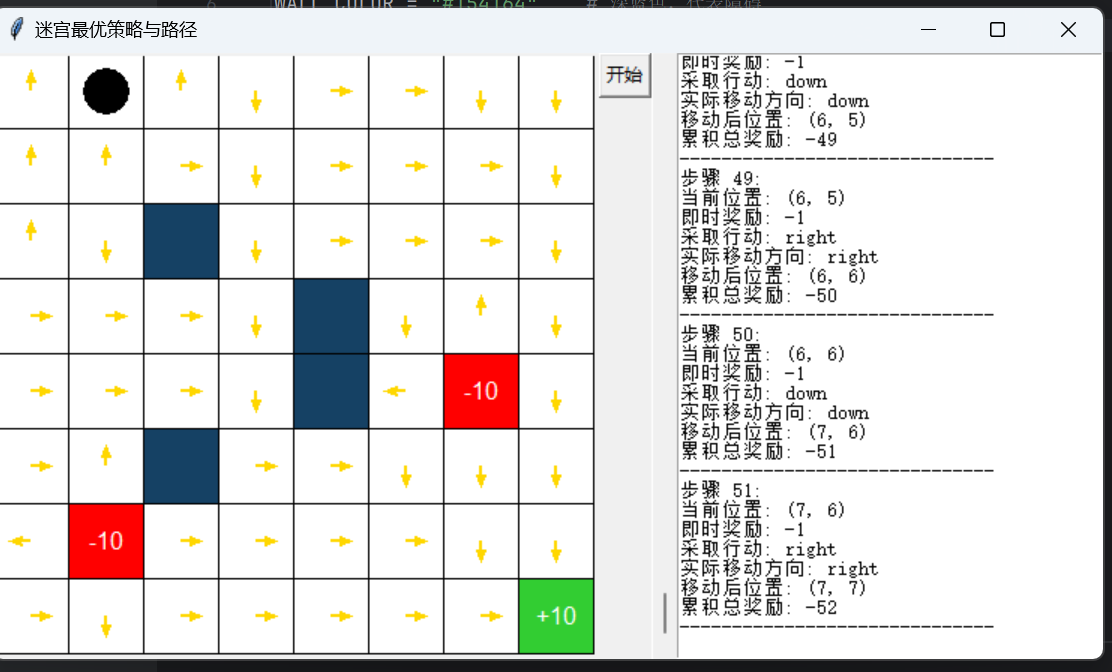


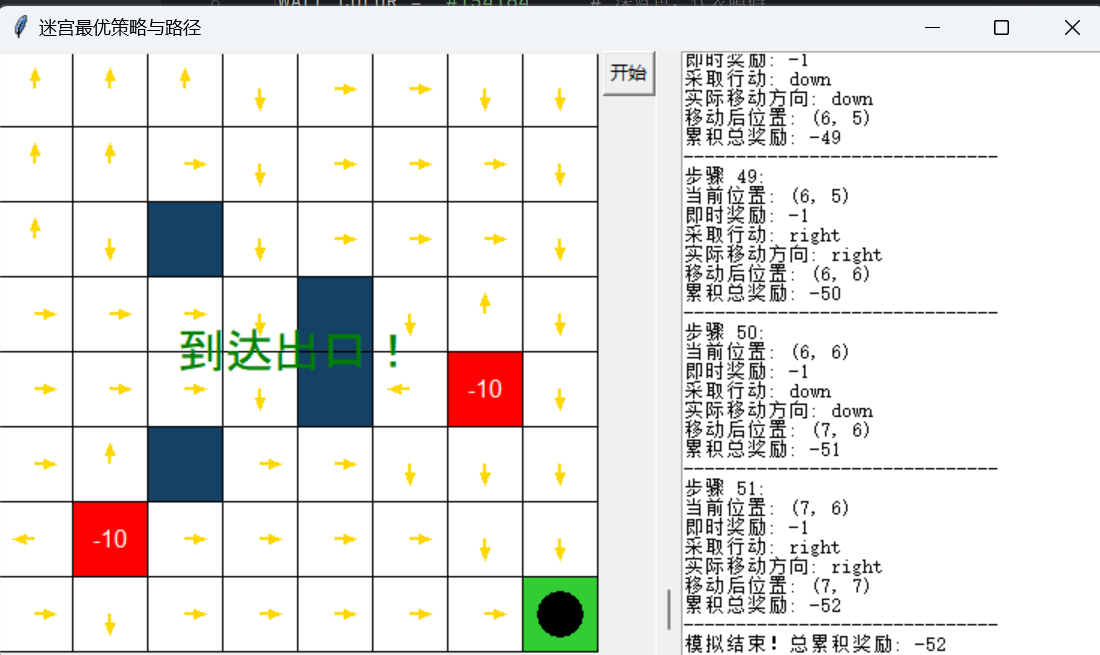
# 二、问题描述与解决方案



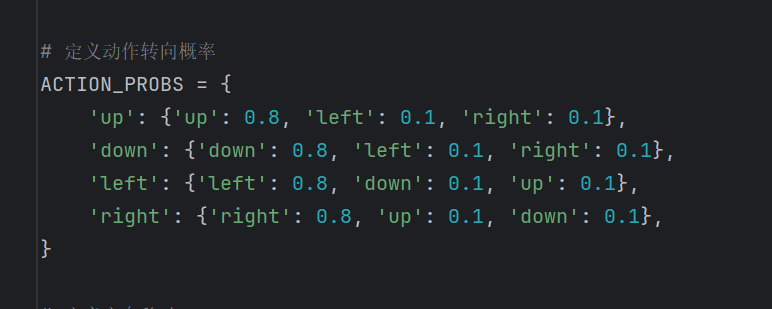
# 三、实验结果

左边显示当前每个格子的最优策略，右边给出MDP迭代过程：

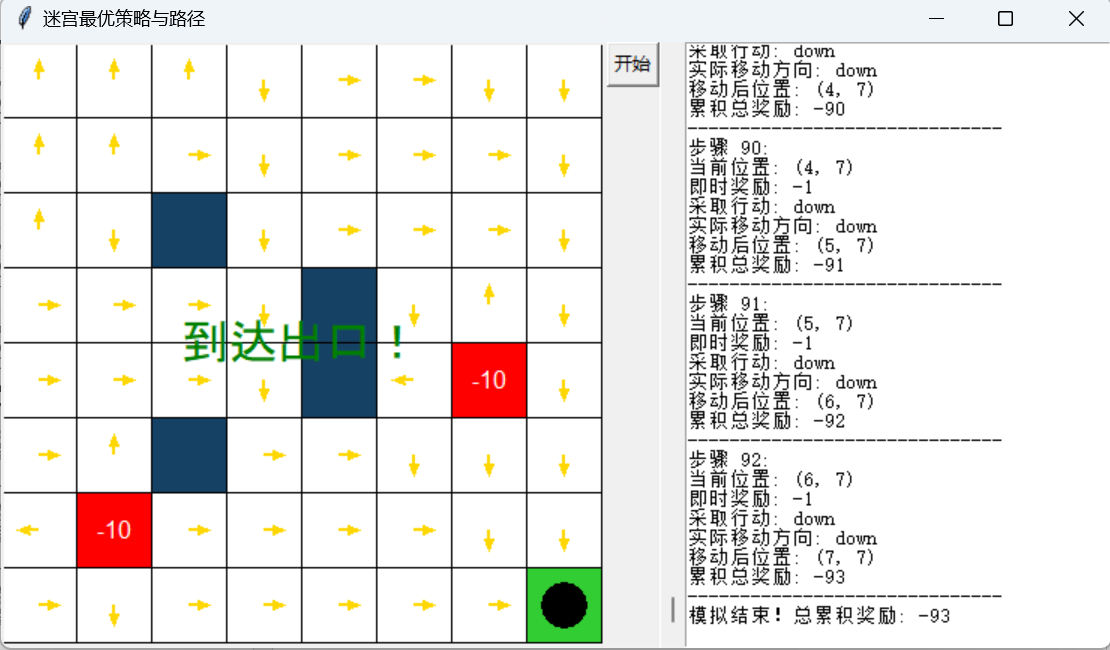


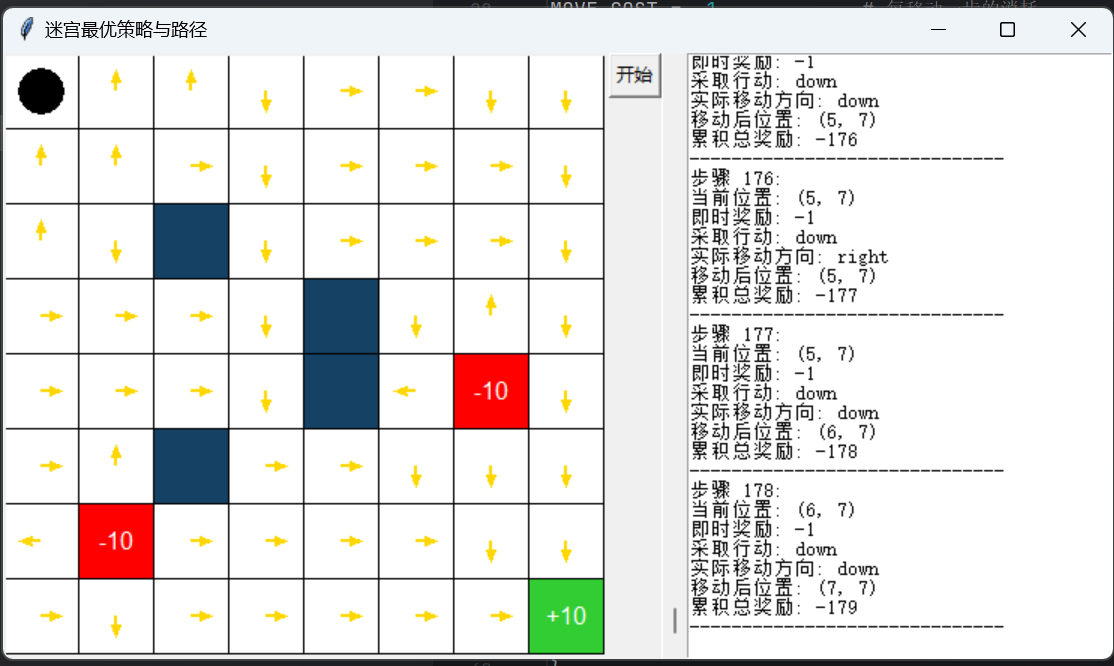


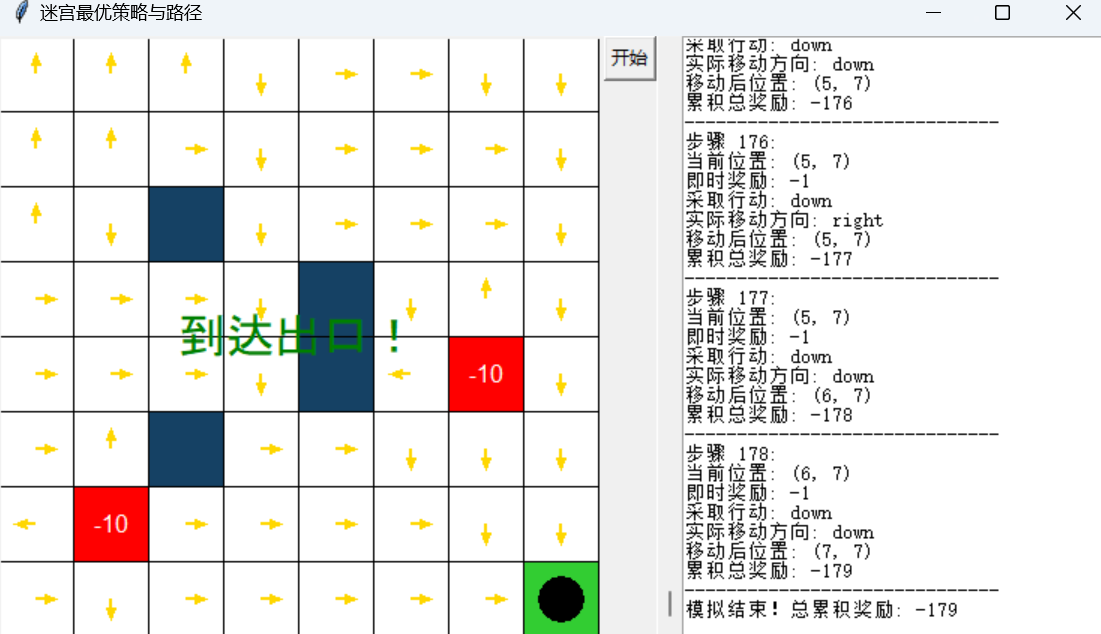
并且由于引入了左右噪声（左右移动中按照个人理解表现为上下噪声，如果需要可按照题意更改概率即可）：

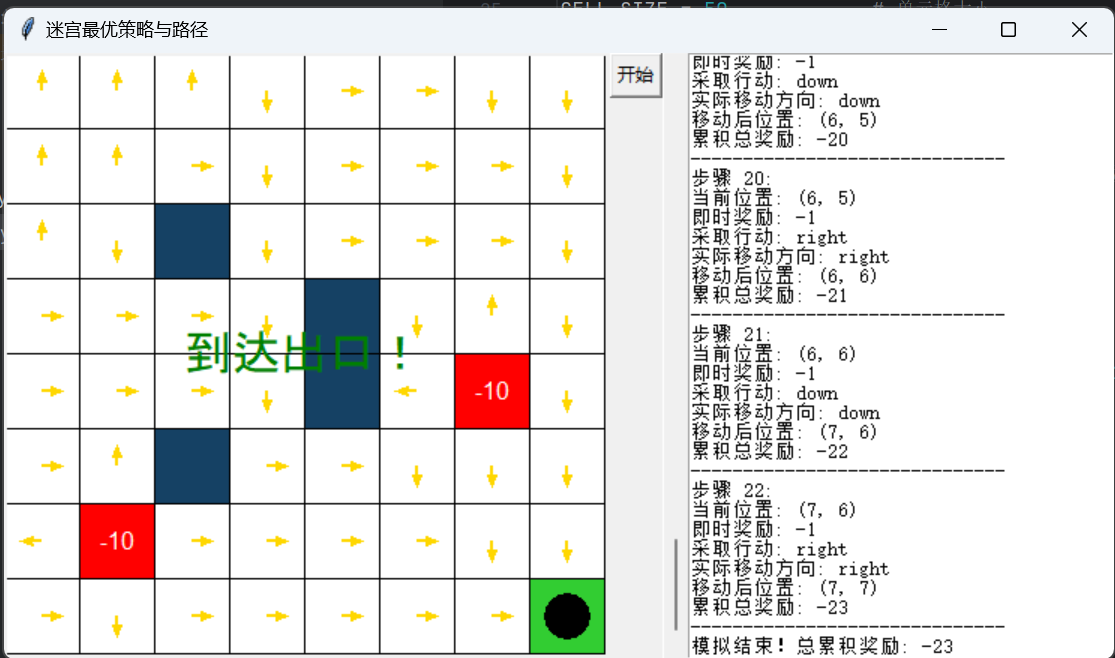


由于左右噪声的存在，每次结果都不一样：









# 四、源码解读

## Problem2.py:

1. **参数定义与设置**
   * 定义了迷宫地图元素颜色、各类参数（折扣因子、噪声、移动消耗等）以及动作转向概率。
2. **地图元素数据与尺寸**
   * MAP\_DATA：一个包含迷宫布局的二维列表。标识各个方块类型（障碍、陷阱、出口、道路、起始位置等）。
   * CELL\_SIZE：单元格大小，用于画布中绘制每个方块的大小。
3. **MDP 类：描述迷宫环境**
   * \_\_init\_\_()：初始化 MDP 对象并解析地图数据。
   * \_parse\_map()：将迷宫地图解析为不同的元素，包括终端状态、障碍、陷阱、出口等。
   * is\_valid\_state()：判断给定位置是否为有效的非障碍位置。
   * get\_possible\_actions()：获取当前状态的所有可能动作。
   * get\_transition\_states\_and\_probs()：基于当前状态和动作，返回可能的下一状态及其概率分布。
   * get\_reward()：返回给定状态的即时奖励。
4. **值迭代算法：求解最优策略**
   * value\_iteration()：基于值迭代的动态规划算法，迭代更新状态价值，直到收敛。在每个状态选择最优动作，输出最终的状态价值和最优策略。
5. **render\_map()：在 Tkinter 画布上渲染地图**
   * 绘制地图中的障碍、陷阱、出口等元素，显示最优策略的箭头方向以及智能体的位置。
6. **simulate\_agent()：模拟智能体的行走路径**
   * 按照最优策略生成智能体的行走路径，并处理噪声效果，记录路径及奖励信息。
   * 输出智能体的每一步操作、即时奖励、累积奖励和位置变化。
7. **main() 函数：主程序逻辑**
   * 创建 MDP 对象并求解最优策略。
   * 创建 Tkinter 窗口及其组件，包括地图画布、输出文本框等。
   * 添加“开始”按钮，点击后执行 start\_animation() 启动智能体的路径模拟。
8. **动画功能：智能体路径展示**
   * animate()：根据模拟路径逐步更新智能体的位置，实时显示其位置变化。模拟结束后显示“到达出口”或“模拟结束”的提示信息。

## 源代码：

import tkinter as tk  
from tkinter import Canvas, Text, Scrollbar, END  
import numpy as np  
  
# 定义颜色  
WALL\_COLOR = "#154164" # 深蓝色，代表障碍  
TRAP\_COLOR = "#FF0000" # 红色，代表陷阱  
EXIT\_COLOR = "#32CD32" # 绿色，代表出口  
ROAD\_COLOR = "#FFFFFF" # 白色，代表可通行道路  
AGENT\_COLOR = "#000000" # 黑色，代表智能体  
POLICY\_COLOR = "#FFD700" # 金色，代表最优策略箭头  
  
# 地图元素定义  
MAP\_DATA = [  
 ["A", "0", "0", "0", "0", "0", "0", "0"],  
 ["0", "0", "0", "0", "0", "0", "0", "0"],  
 ["0", "0", "1", "0", "0", "0", "0", "0"],  
 ["0", "0", "0", "0", "1", "0", "0", "0"],  
 ["0", "0", "0", "0", "1", "0", "-10", "0"],  
 ["0", "0", "1", "0", "0", "0", "0", "0"],  
 ["0", "-10", "0", "0", "0", "0", "0", "0"],  
 ["0", "0", "0", "0", "0", "0", "0", "+10"]  
]  
  
CELL\_SIZE = 50 # 单元格大小  
DISCOUNT\_FACTOR = 0.5 # 折扣因子  
NOISE = 0.2 # 噪声  
MOVE\_COST = -1 # 每移动一步的消耗  
  
ACTION\_DELAY = 500 # 动作之间的时间间隔，单位为毫秒  
  
ACTIONS = ['up', 'down', 'left', 'right']  
  
# 定义动作转向概率  
ACTION\_PROBS = {  
 'up': {'up': 0.8, 'left': 0.1, 'right': 0.1},  
 'down': {'down': 0.8, 'left': 0.1, 'right': 0.1},  
 'left': {'left': 0.8, 'down': 0.1, 'up': 0.1},  
 'right': {'right': 0.8, 'up': 0.1, 'down': 0.1},  
}  
  
# 定义方向移动  
DELTA = {  
 'up': (-1, 0),  
 'down': (1, 0),  
 'left': (0, -1),  
 'right': (0, 1),  
}  
  
class MDP:  
 def \_\_init\_\_(self, map\_data):  
 self.map\_data = map\_data  
 self.rows = len(map\_data)  
 self.cols = len(map\_data[0])  
 self.states = []  
 self.rewards = {}  
 self.terminal\_states = []  
 self.obstacles = []  
 self.traps = []  
 self.exit = None  
 self.agent\_start = None  
 self.\_parse\_map()  
  
 def \_parse\_map(self):  
 for i in range(self.rows):  
 for j in range(self.cols):  
 cell = self.map\_data[i][j]  
 if cell == '1':  
 self.obstacles.append((i, j))  
 elif cell == '-10':  
 self.states.append((i, j))  
 self.rewards[(i, j)] = -10  
 self.traps.append((i, j))  
 # 不将陷阱添加到终端状态  
 elif cell == '+10':  
 self.states.append((i, j))  
 self.rewards[(i, j)] = 10  
 self.exit = (i, j)  
 self.terminal\_states.append((i, j)) # 仅将出口作为终端状态  
 elif cell == 'A':  
 self.states.append((i, j))  
 self.rewards[(i, j)] = MOVE\_COST  
 self.agent\_start = (i, j)  
 else: # '0'  
 self.states.append((i, j))  
 self.rewards[(i, j)] = MOVE\_COST  
  
 def is\_valid\_state(self, state):  
 i, j = state  
 return 0 <= i < self.rows and 0 <= j < self.cols and state not in self.obstacles  
  
 def get\_possible\_actions(self, state):  
 if state in self.terminal\_states:  
 return []  
 else:  
 return ACTIONS  
  
 def get\_transition\_states\_and\_probs(self, state, action):  
 transition\_probs = {}  
 for a, prob in ACTION\_PROBS[action].items():  
 new\_state = (state[0] + DELTA[a][0], state[1] + DELTA[a][1])  
 if not self.is\_valid\_state(new\_state):  
 new\_state = state # 碰壁，留在原地  
 if new\_state in transition\_probs:  
 transition\_probs[new\_state] += prob  
 else:  
 transition\_probs[new\_state] = prob  
 result = [(new\_state, prob) for new\_state, prob in transition\_probs.items()]  
 return result  
  
 def get\_reward(self, state):  
 return self.rewards.get(state, MOVE\_COST)  
  
def value\_iteration(mdp, epsilon=0.01):  
 V = {}  
 for state in mdp.states:  
 V[state] = 0 # 初始化价值函数  
  
 while True:  
 delta = 0  
 V\_old = V.copy()  
 for state in mdp.states:  
 if state in mdp.terminal\_states:  
 V[state] = mdp.get\_reward(state)  
 continue  
 max\_value = float('-inf')  
 for action in mdp.get\_possible\_actions(state):  
 value = 0  
 for next\_state, prob in mdp.get\_transition\_states\_and\_probs(state, action):  
 reward = mdp.get\_reward(next\_state)  
 value += prob \* (reward + DISCOUNT\_FACTOR \* V\_old[next\_state])  
 if value > max\_value:  
 max\_value = value  
 delta = max(delta, abs(V[state] - max\_value))  
 V[state] = max\_value  
 if delta < epsilon:  
 break  
  
 # 提取最优策略  
 policy = {}  
 for state in mdp.states:  
 if state in mdp.terminal\_states:  
 policy[state] = None  
 continue  
 max\_value = float('-inf')  
 best\_action = None  
 for action in mdp.get\_possible\_actions(state):  
 value = 0  
 for next\_state, prob in mdp.get\_transition\_states\_and\_probs(state, action):  
 reward = mdp.get\_reward(next\_state)  
 value += prob \* (reward + DISCOUNT\_FACTOR \* V[next\_state])  
 if value > max\_value:  
 max\_value = value  
 best\_action = action  
 policy[state] = best\_action  
  
 return V, policy  
  
def render\_map(canvas, map\_data, agent\_position=None, policy=None):  
 *"""在Tkinter画布上渲染地图"""* canvas.delete("all") # 清除画布内容  
 for i, row in enumerate(map\_data):  
 for j, cell in enumerate(row):  
 x1, y1 = j \* CELL\_SIZE, i \* CELL\_SIZE  
 x2, y2 = x1 + CELL\_SIZE, y1 + CELL\_SIZE  
  
 if cell == "1": # 障碍  
 color = WALL\_COLOR  
 elif cell == "0": # 可通行道路  
 color = ROAD\_COLOR  
 elif cell == "-10": # 陷阱  
 color = TRAP\_COLOR  
 canvas.create\_rectangle(x1, y1, x2, y2, fill=color, outline="black")  
 canvas.create\_text((x1 + x2) // 2, (y1 + y2) // 2, text="-10", fill="white", font=("Arial", 12))  
 continue  
 elif cell == "+10": # 出口  
 color = EXIT\_COLOR  
 canvas.create\_rectangle(x1, y1, x2, y2, fill=color, outline="black")  
 canvas.create\_text((x1 + x2) // 2, (y1 + y2) // 2, text="+10", fill="white", font=("Arial", 12))  
 continue  
 elif cell == "A": # 起始位置  
 color = ROAD\_COLOR  
 else:  
 color = ROAD\_COLOR  
  
 # 绘制方块  
 canvas.create\_rectangle(x1, y1, x2, y2, fill=color, outline="black")  
  
 # 绘制最优策略  
 if policy and (i, j) in policy and policy[(i, j)] is not None:  
 action = policy[(i, j)]  
 x, y = j \* CELL\_SIZE + CELL\_SIZE // 2, i \* CELL\_SIZE + CELL\_SIZE // 2  
 if action == 'up':  
 dx, dy = 0, -15  
 elif action == 'down':  
 dx, dy = 0, 15  
 elif action == 'left':  
 dx, dy = -15, 0  
 elif action == 'right':  
 dx, dy = 15, 0  
 canvas.create\_line(x, y, x + dx, y + dy, arrow=tk.LAST, fill=POLICY\_COLOR, width=2)  
  
 # 绘制智能体  
 if agent\_position:  
 i, j = agent\_position  
 x1, y1 = j \* CELL\_SIZE, i \* CELL\_SIZE  
 x2, y2 = x1 + CELL\_SIZE, y1 + CELL\_SIZE  
 canvas.create\_oval(x1 + 10, y1 + 10, x2 - 10, y2 - 10, fill=AGENT\_COLOR)  
  
def simulate\_agent(mdp, policy, output\_text):  
 *"""生成智能体按照最优策略的行走路径，并输出详细信息"""* state = mdp.agent\_start  
 path = [state]  
 total\_reward = 0 # 记录总奖励  
 step = 0 # 记录步数  
  
 while state != mdp.exit:  
 action = policy[state]  
 if action is None:  
 break  
  
 # 输出当前状态和行动  
 reward = mdp.get\_reward(state)  
 total\_reward += reward  
  
 output\_info = f"步骤 {step}:\n"  
 output\_info += f"当前位置: {state}\n"  
 output\_info += f"即时奖励: {reward}\n"  
 output\_info += f"采取行动: {action}\n"  
  
 # 由于有噪声，这里模拟一次实际动作  
 transition\_probs = {}  
 for a, prob in ACTION\_PROBS[action].items():  
 next\_state = (state[0] + DELTA[a][0], state[1] + DELTA[a][1])  
 if not mdp.is\_valid\_state(next\_state):  
 next\_state = state  
 if next\_state in transition\_probs:  
 transition\_probs[next\_state] += prob  
 else:  
 transition\_probs[next\_state] = prob  
 next\_states = list(transition\_probs.keys())  
 probs = [transition\_probs[s] for s in next\_states]  
 # 根据概率选择下一个状态  
 idx = np.random.choice(range(len(next\_states)), p=probs)  
 next\_state = next\_states[idx]  
  
 # 输出实际移动结果  
 actual\_action = None  
 for a in ACTIONS:  
 ns = (state[0] + DELTA[a][0], state[1] + DELTA[a][1])  
 if not mdp.is\_valid\_state(ns):  
 ns = state  
 if ns == next\_state:  
 actual\_action = a  
 break  
  
 output\_info += f"实际移动方向: {actual\_action}\n"  
 state = next\_state  
 path.append(state)  
 step += 1  
  
 output\_info += f"移动后位置: {state}\n"  
 output\_info += f"累积总奖励: {total\_reward}\n"  
 output\_info += "-" \* 30 + "\n"  
  
 # 在文本区域输出信息  
 output\_text.insert(END, output\_info)  
 output\_text.see(END) # 滚动到最后一行  
  
 if len(path) > 500: # 防止无限循环  
 break  
 return path, total\_reward  
  
def main():  
 # 创建MDP  
 mdp = MDP(MAP\_DATA)  
 # 值迭代求解最优策略  
 V, policy = value\_iteration(mdp)  
 # 创建Tkinter主窗口  
 root = tk.Tk()  
 root.title("迷宫最优策略与路径")  
  
 # 创建画布  
 canvas = Canvas(root, width=mdp.cols \* CELL\_SIZE, height=mdp.rows \* CELL\_SIZE)  
 canvas.pack(side=tk.LEFT)  
  
 # 创建文本区域用于输出信息  
 output\_text = Text(root, width=40)  
 output\_text.pack(side=tk.RIGHT, fill=tk.Y)  
 scrollbar = Scrollbar(root)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 output\_text.config(yscrollcommand=scrollbar.set)  
 scrollbar.config(command=output\_text.yview)  
  
 # 渲染初始地图和最优策略  
 render\_map(canvas, MAP\_DATA, agent\_position=mdp.agent\_start, policy=policy)  
  
 # 在外层作用域中初始化变量  
 path = []  
 total\_reward = 0  
  
 # 添加“开始”按钮  
 def start\_animation():  
 nonlocal path, total\_reward  
 # 生成智能体的路径，并输出详细信息  
 path, total\_reward = simulate\_agent(mdp, policy, output\_text)  
 animate()  
  
 # 开始动画  
 def animate(step=0):  
 if step < len(path):  
 agent\_position = path[step]  
 render\_map(canvas, MAP\_DATA, agent\_position=agent\_position, policy=policy)  
 root.after(ACTION\_DELAY, animate, step + 1)  
 else:  
 # 动画结束，显示结束信息  
 state = path[-1]  
 if state == mdp.exit:  
 canvas.create\_text(mdp.cols \* CELL\_SIZE // 2, mdp.rows \* CELL\_SIZE // 2,  
 text="到达出口！", fill="green", font=("Arial", 24))  
 else:  
 canvas.create\_text(mdp.cols \* CELL\_SIZE // 2, mdp.rows \* CELL\_SIZE // 2,  
 text="模拟结束", fill="blue", font=("Arial", 24))  
 # 输出总奖励  
 output\_text.insert(END, f"模拟结束！总累积奖励: {total\_reward}\n")  
  
 start\_button = tk.Button(root, text="开始", command=start\_animation)  
 start\_button.pack()  
  
 # 运行主循环  
 root.mainloop()  
  
  
# 运行程序  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()