南開大學
Nankai University

人 工 智 能 技 术 实 验
实 验 报 告

实验名称：A*算法的实现

姓　　名：刘崇轩

学　　号：2312801

专　　业：智能科学与技术

人工智能学院

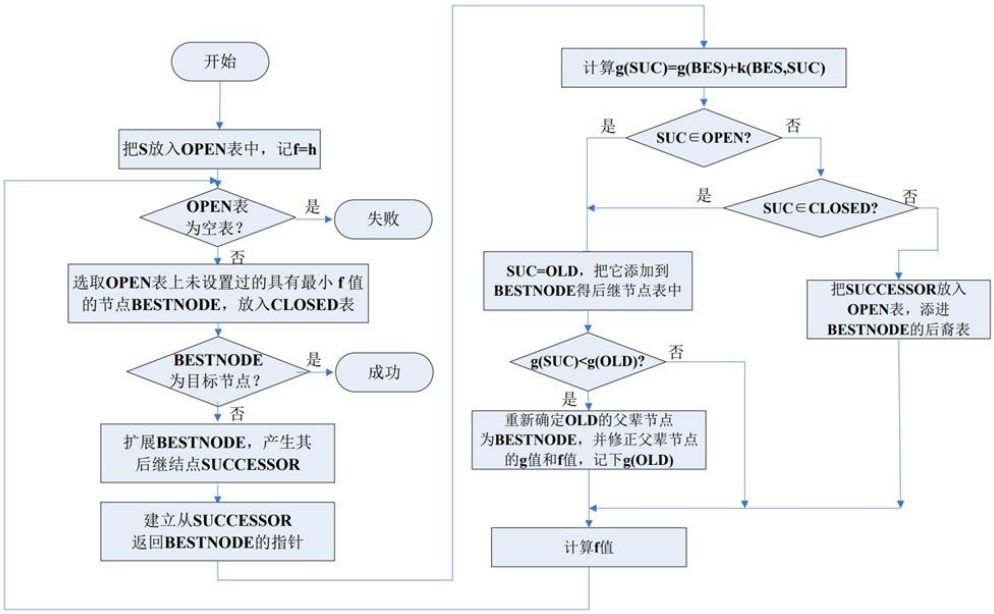
2024 年 9 月

一、 问题简述

A*算法一种典型的启发式搜索算法，是静态路网中求解最短路径最有效的方法之一。启发式搜索又称为有信息搜索，它是利用问题拥有的启发信息来引导搜索，达到减少搜索范围、降低问题复杂度的目的，通常拥有更好的性能。A*算法关键部分为启发函数：

$$F = G + H$$

其中，G 为从起点 A 移动到目前方格的代价，H 是该方格到终点 B 的估算成本，F 为寻找下一个遍历节点的依据。



伪代码：

初始化 open_list 和 close_list；

* 将起点加入 open_list 中，并设其移动代价为 0；

* 如果 open_list 不为空，则从 open_list 中选取移动代价最小的节点 n：

    * 如果节点 n 为终点，则：

        * 从终点开始逐步追踪 parent 节点，一直达到起点；

        * 返回找到的结果路径，算法结束；

    * 如果节点 n 不是终点，则：

        * 将节点 n 从 open_list 中移除，并加入 close_list 中；

        * 遍历节点 n 所有的邻近节点：

            * 如果邻近节点 m 在 close_list 中，则：

                * 跳过，选取下一个邻近节点

            * 如果邻近节点 m 在 open_list 中，则：

                * 计算起点经 n 到 m 的 g 值

* 如果此时计算出来的 g 值小于原来起点经 m 的原父节点到 m 的 g 值：

* 更新 m 的父节点为 n，重新计算 m 的移动代价 f=g+h

* 否则，跳过

* 如果邻近节点 m 不在 open_list 也不在 close_list 中，则：

* 设置节点 m 的 parent 为节点 n

* 计算节点 m 的移动代价 f=g+h

* 将节点 m 加入 open_list 中

二、 实验目的

1. 掌握图搜索算法思路和流程，理解无信息搜索与有信息搜索的区别；
2. 对于启发式搜索，理解 A*算法估值函数的选取对算法性能的影响；
3. 对于启发式搜索，理解 A*算法求解流程和搜索顺序；
4. 深入理解对图形化界面设计。（选做）

三、 实验内容

1. 实现 Dijkstra 算法（H=0），以带有障碍物的二维地图为基础，寻找到达目标点的最短路径；
2. 实现 A*算法，以带有障碍物的二维地图为基础，寻找到达目标点的最短路径；
3. 对 Dijkstra 算法和 A*算法的性能进行比较分析；
4. 对实验进行图形化界面设计，展示搜索过程和最优路径。（选做）

四、 编译环境

1. 操作系统：Windows 11
2. Python：3.12
3. 主要第三方库：matplotlib 3.5+、pillow (PIL) 9.0+、numpy 1.22+、tkinter
4. 在 PyCharm Community Edition 下编译

五、 实验步骤（包括重要程序代码解释）

1. 启动脚本 dijkstra&astar.py。
2. 在界面左上角输入起点 (x,y) 和终点 (x,y)。脚本中已设默认值（1,1 到 7,7）。
3. 点击 Run：脚本构建地图、调用 a_start_search 并把每个步骤生成的 matplotlib Figure 存入全局 openc，最后生成 success（最终路径图）。
4. 点击 Next：逐帧显示搜索过程（依次显示 openc 中帧），播放完以后显示最终路径 success。

程序代码解释：（完整程序见 dijkstra&astar.py）

1. **draw_node_ax(maze, ax)**
   (1) 将 maze 中值为 1 的格子绘制为矩形障碍（使用 matplotlib.patches.Rectangle）。
   (2) 注意：绘制时把数组坐标转为图像坐标，(j, n-1-i) 用于把二维数组的(行,列)转成左下为原点的轴上合适位置。
2. **make_frame(step_index, start, end, next_list, already_list, now_grid=None, annotate_all=True)**
   (1) 为每一个 A*步骤创建一个 matplotlib.Figure:
   ① 画格子网格、障碍;
   ② 在画布上绘制 open 集合（next_list）和 closed 集合（already_list）的点，标注每个节点的 f/g/h 值（如果 annotate_all 为真）;
   ③ 将当前处理节点 now_grid 用更大的标记突出显示;
   ④ 绘制起点和终点的标注（START / END）和它们的 f/g/h 值。
   (2) 返回 fig（并不直接显示），调用方会把 fig 存到 openc 列表中，供 UI 按帧显示。
3. **a_start_search(start, end)**
   (1) 主体 A* 搜索循环:
   ① 初始化 next_list（open）和 already_list（closed），将起点放入 next_list;
   ② 每次循环：用 find_min_grid 从 next_list 选出 f 最小的节点 now_grid；将它从 next 中转到 already 中；找到它的邻居并根据条件把新的邻居加入 next 中（并调用 Grid.init_grid 计算 g/h/f）;
   ③ 搜到目标则返回目标 Grid（其 parent 指向路径上的节点），找不到则返回 None。
   (2) 注意（实现细节）：函数里在每次循环开始都调用 make_frame 并把返回的 Figure 存入全局 openc，从而保存逐步可视化帧。
4. **find_neighbors(grid, next_list, already_list)**
   （1）检查 8 个方向是否有效（用 is_valid_grid），若有效就创建 Grid 对象加入返回列表。
   （2）还包含对已在 next 或 already 中的邻居的 g 值比较与更新(若当前路径更短则替换/更新）；对直角与斜向移动使用不同的代价（直角+10，斜向+14）。
   （3）额外包含“禁止斜向穿角”的逻辑（如果两侧格子都是障碍，则禁止斜向移动）——在主循环里也有类似判断以避免切角。
5. **is_valid_grid(x,y, next_list, already_list)**
   边界检查、是否为障碍（MAZE[x][y] == 1）以及是否已在 open/closed 中，三项之一不满足即返回 False。
6. **Grid 类与 init_grid**
   (1) Grid 保存 x,y,f,g,h,parent。
   (2) init_grid(parent, end)：根据与父节点的曼哈顿关系设置 g（直角+10、斜向+14），计算启发 h（曼哈顿距离 * 10），然后 f=g+h。
7. **draw_picture(m_arg, n_arg, path)**

绘制最终路径（点与连线）并重绘障碍与网格。path 为反向链表，调用处先把 result_grid 沿 parent 回溯生成 path。

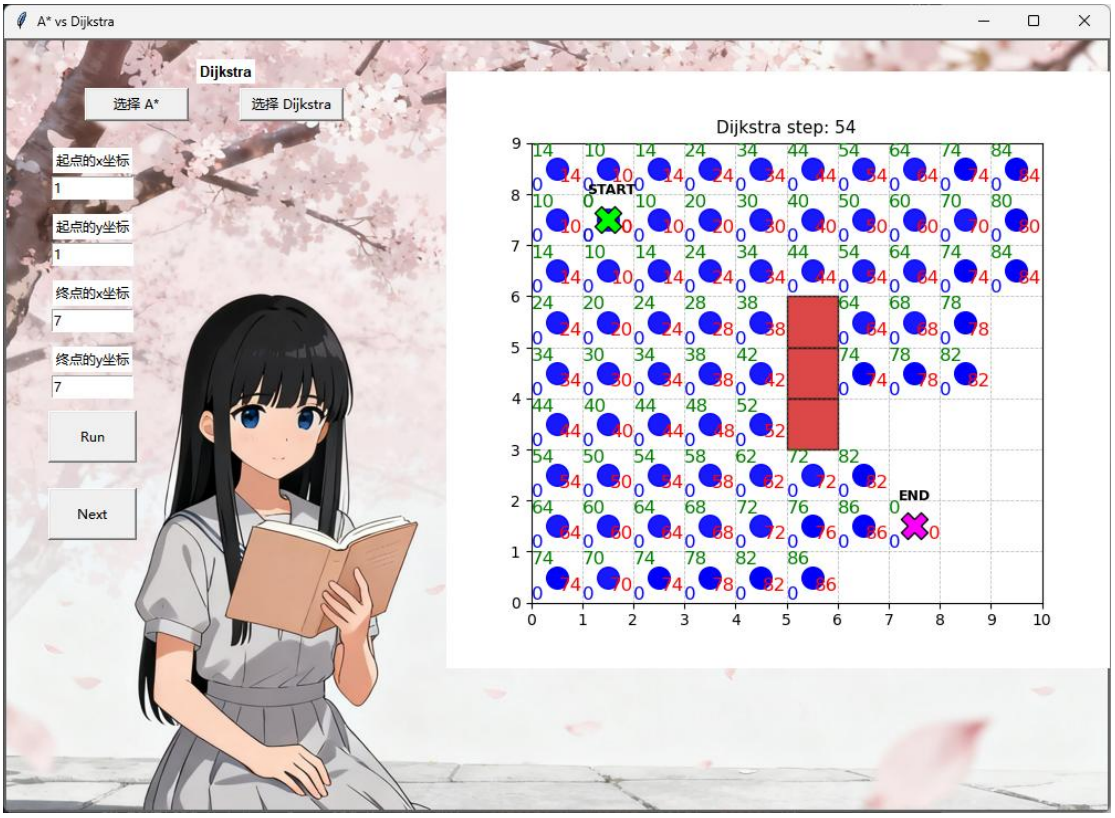
8. **Tkinter UI 部分（moveot, moveot1）**

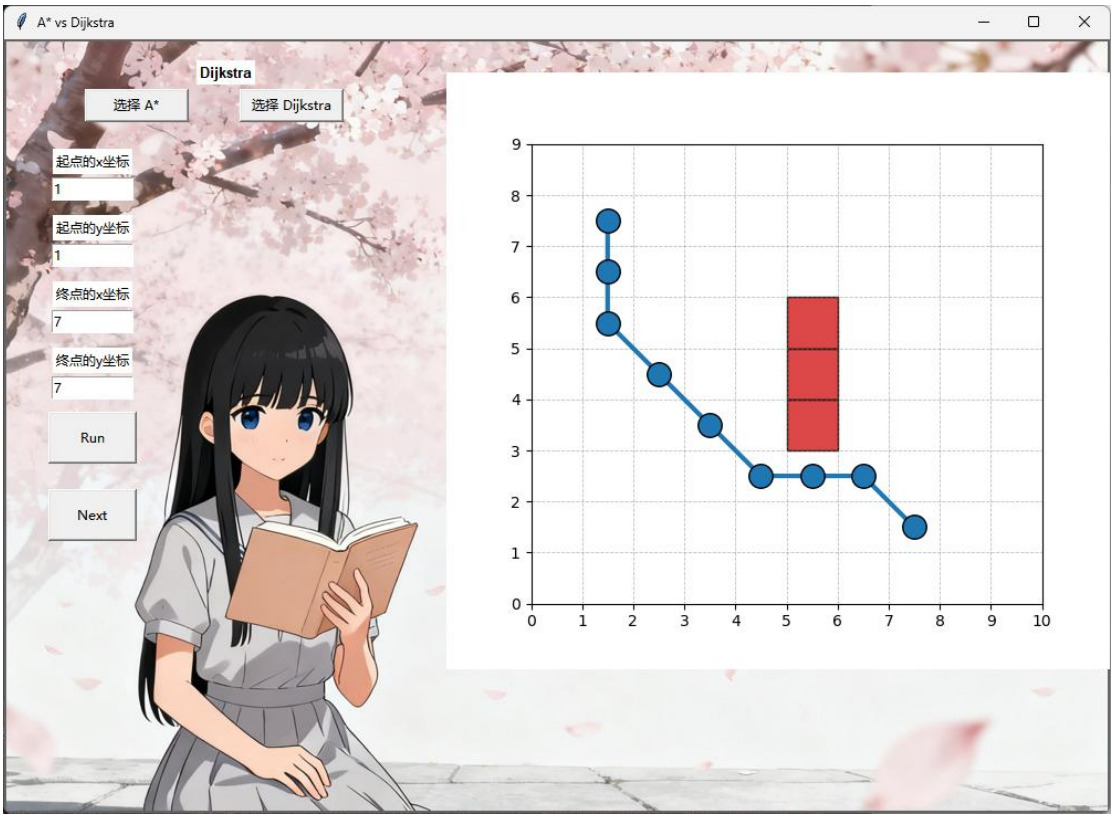
(1) moveot()：读入四个输入框的起终点，构造 MAZE（代码中硬编码了一个带竖条障碍的示例），设定 m,n，然后调用 a_start_search 获取 result_grid，并生成 path，最后调用 draw_picture 生成 success 图。并把 openc（逐步帧）和 success（最终路径）准备好供显示。

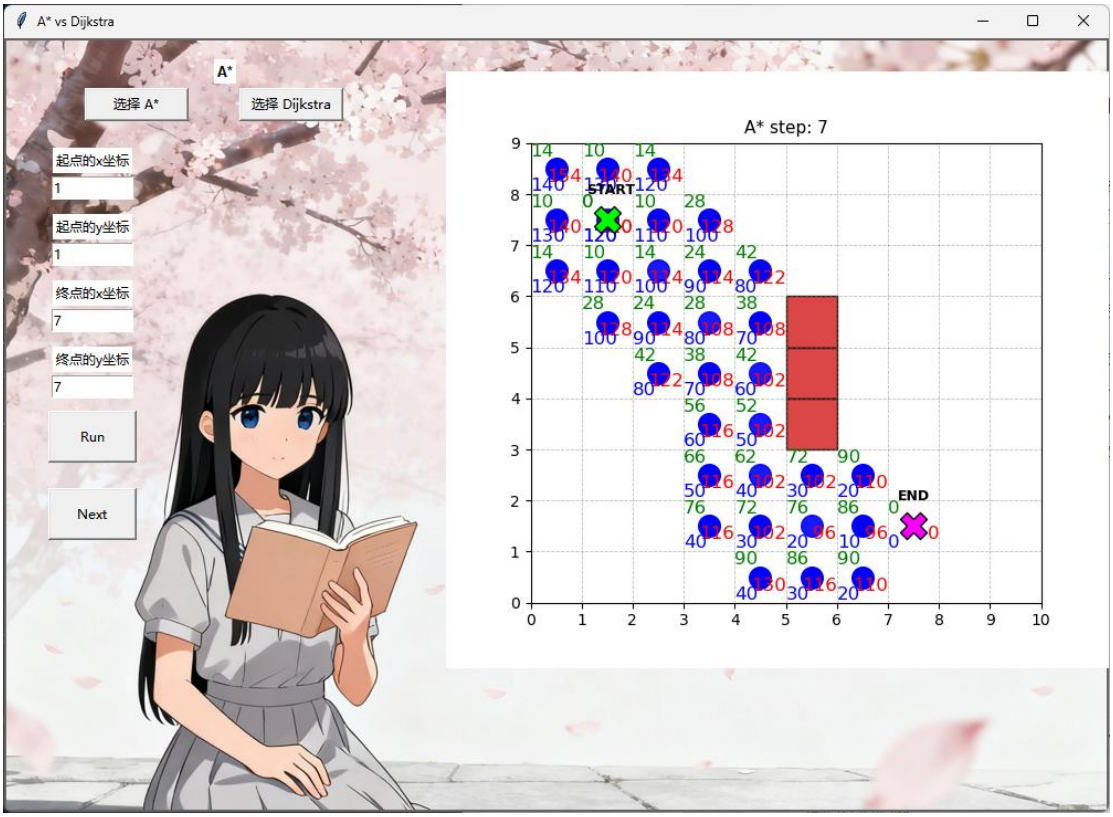
(2) moveot1()：每次按 Next 显示 openc[count] 中第 count 帧；当 count 超过 openc 长度时显示 success（最终路径）。

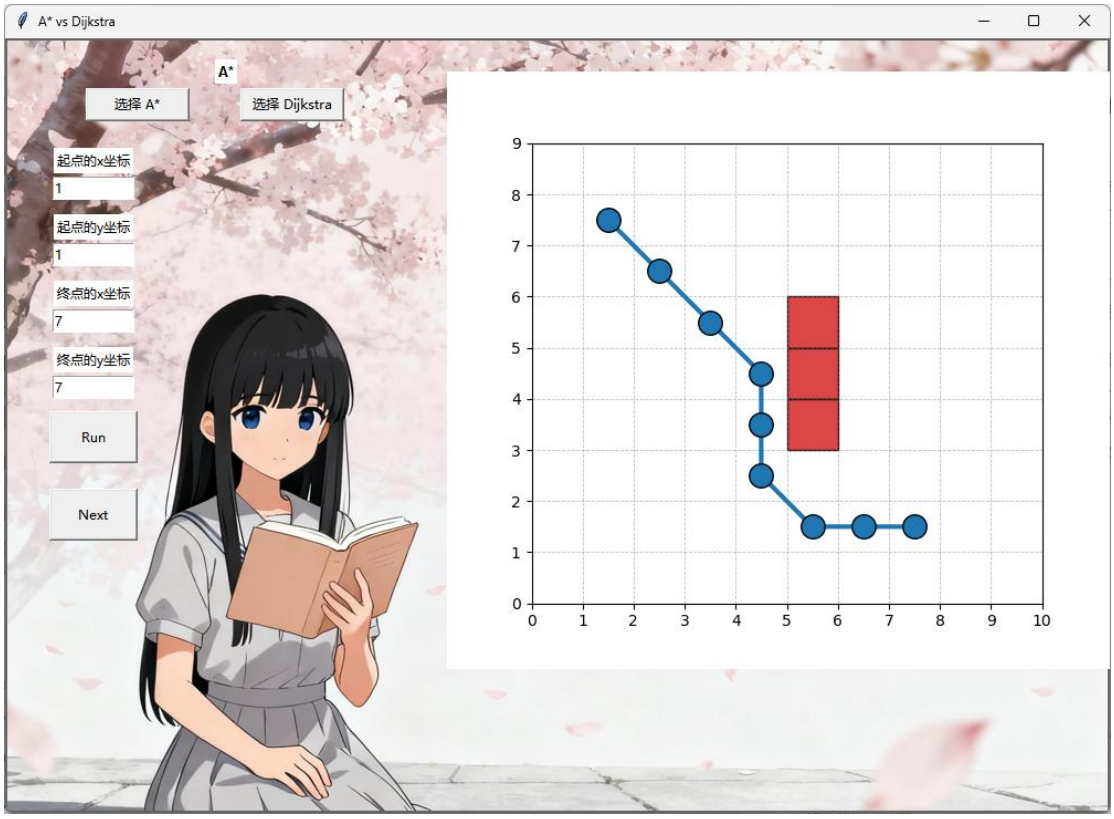
(3) UI 布局使用 Canvas 放背景图，并把按钮/Entry 放到 bg_canvas 上。

六、 实验结果




A* vs Dijkstra
Dijkstra
选择 A*
选择 Dijkstra
起点的x坐标
1
起点的y坐标
1
终点的x坐标
7
终点的y坐标
7
Run
Next


A* vs Dijkstra
A*
选择 A*
选择 Dijkstra
起点的x坐标
1
起点的y坐标
1
终点的x坐标
7
终点的y坐标
7
Run
Next
A* step: 7
START
END



七、 分析总结

本次实验将 A* 与 Dijkstra 并列实现并可视化，方便直观比较两者在相同地图与起终点条件下的搜索行为与性能差异。实验关键结论如下：

（1）效率差异：A* 在使用有效启发式的情况下，通常比 Dijkstra 更高效——它把搜索聚焦到有希望到达目标的方向，从而显著减少被扩展的节点数和搜索步数；Dijkstra 因为没有启发信息，会均匀展开波前，扩展更多节点、花费更多时间。

（2）启发式的正确性至关重要：当前代码使用的启发函数为 h = (|dx|+|dy|)*10（曼哈顿距离×10）。当允许 8 向移动且斜向代价为 14 时，曼哈顿启发对斜向移动会高估真实最短代价（例如从(0,0)到(1,1)的曼哈顿 h=20，但对角实际最小代价是 14），因此它不是对 8-连通且允许斜移的场景的可采纳启发式，可能导致 A* 找到非最优路径或不满足最优性保证。