人工智能导论

第一次大作业

实验报告

自 66 韩凯乾 2016010951

一、项目概述与编译环境

本次大作业选题为题目 2，即小兔子找胡萝卜的迷宫问题，最终完成开发的游戏名为：Caveman and Treasure,即：穴居人寻宝，游戏整体界面如下：

该项目在 windows 下编译通过，所需环境为 python3，编写 GUI 所用的库为 pygame，在运行作业前，需要配置依赖项，即在 main.py 的路径下打开 cmd，并运行：

pip install –r requirement.txt

配置完依赖项后即可运行游戏：

python main.py

为了方便测试不同搜索算法的效率，编写了脚本 test.py 进行测试：

python test.py --maze_size 10 (设置为需要的迷宫大小，建议为 5-25，否则可

能超过递归上界)

若文件上传中存在偏差，可以从

https://github.com/hanpig1998/AI-Introduction-project-1 中下载

或在已经安装 git 的电脑中执行：

git clone https://github.com/hanpig1998/AI-Introduction-project-1.git

二、问题的数学建模

由于迷宫的实质为一个由 0,1 构成的矩阵

其中 1 代表可行走的区域，0 代表障碍物

由于穴居人生活区域中存在冰面，因此设置了随机道路中存在冰块，在冰块上穴居人会打滑，导致行走的代价翻倍。

在本问题中，设置穴居人初始位置处的值为 10，宝藏处的值为 1，则迷宫求解转化为该矩阵中 10 处到 1 处的带权连通路径。

三、算法实现

1. 迷宫的创建

事实上，本次大作业中的一大难点在于迷宫如何创建，在实现过程中进行了如下尝试：

(1) 对每一小格随机添加障碍物

(2) 限定障碍物的形状，并在地图中随机放置障碍物，如(L 型，H 型)

但实际的创建结果均不理想(迷宫的样子不像迷宫：出现大量空白/障碍堆积现象)

最后经过查阅资料，采取递归回溯的算法生成迷宫，算法如下：

①每次把新找到的未访问迷宫单元作为优先

②寻找其相邻的未访问过的迷宫单元，直到所有的单元都被访问到

通俗的说，就是从起点开始随机走，走不通了就返回上一步，从下一个能走

的地方再开始随机走。

在创建完迷宫后，再在迷宫中随机指定人物的初始位置与宝藏，以及随机冰面的位置。

该迷宫创建算法可以创建不同长、宽、障碍物、初始\终点位置，具体代码详见：maze.py

2.搜索算法描述

本次实验中采取了四种搜索算法进行求解，实现方式均为迭代：

(1)深度优先搜索

利用 stack 实现，先让起点入栈，之后进行如下迭代

①栈弹出顶点 I，并标记 I 为已访问的

②检查 I 是否为宝藏，若为宝藏，迭代结束

③依次检查 I 的领域，将其中未访问过的顶点入栈

(2)宽度优先搜索

利用 queue 实现，先让起点入列，之后进行如下迭代

①队列出列顶点 I，并标记 I 为已访问的

②检查 I 是否为宝藏，若为宝藏，迭代结束

③依次检查 I 的领域，将其中未访问过的顶点入队列

(3)一致代价搜索

利用 PriorityQueue 实现，实现 Maze_unit 类对顶点进行包装，并重新定义优先级队列的排序方法，在本次实现的一致代价搜索中，h(n)为路径代价之和，g(n)为 0

先让起点入列，之后进行如下迭代

①优先级队列出列顶点 I，并标记 I 为已访问的

②检查 I 是否为宝藏，若为宝藏，迭代结束

③依次检查 I 的领域，将其中未访问过的顶点入优先级队列

由于代价为路径长度，因此在本项目中，一致代价搜索的结果与宽度优先搜索基本一致。

(4)A*搜索

利用 PriorityQueue 实现，实现 Maze_unit 类对顶点进行包装，并重新定义优先级队列的排序方法，在本次实现的一致代价搜索中，h(n)为路径代价之和，g(n)为顶点距离宝藏的曼哈顿距离乘以相关系数

先让起点入列，之后进行如下迭代

①优先级队列出列顶点 I，并标记 I 为已访问的

②检查 I 是否为宝藏，若为宝藏，迭代结束

③依次检查 I 的领域，将其中未访问过的顶点入优先级队列

搜索算法的代码详见:solution.py

四、项目架构与 GUI 设计

(1)项目架构

本次项目将核心部分与 GUI 部分分离，各 py 文件内容如下：

maze:迷宫创建

solution:问题求解

people: GUI 中的类：人物

wall:GUI 中的类:墙壁

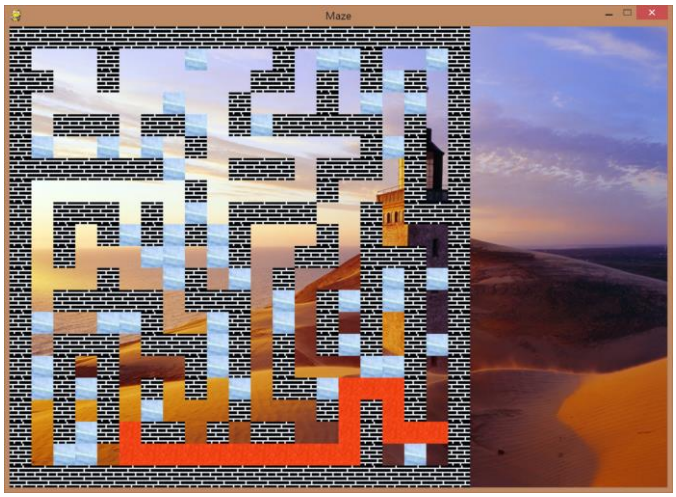
main:主函数，基于 pygame 的 GUI 编写

test:测试脚本，直接统计各算法 1000 次所需的时间与扩展节点数

(2)GUI 设计

为了方便运行与直观观看搜索过程，

在迷宫游戏的右侧有五个按键，利用鼠标点击，可以分别观看不同搜索算法的搜索过程，点击 View path 可以观看利用 A*搜索得到的通向终点的最短路径，此外玩家可以直接使用 wsad 按键控制上下左右进行游戏。若不想手动控制抵达终点，也可以直接选中 skip 按钮跳过该关卡。



View path

GUI 画面设计：

GUI 的素材均位于子文件夹 images 中，其中人物行走过程的上下左右利用数字图像处理的方法进行了不同对待：

同时您也可以在游戏时打开系统的声音，感受 bgm 带来的不同体验

Bgm：music.ogg

五、搜索算法效率对比

利用 test.py 对不同的搜索算法效率进行了对比如下：

迷宫大小：7*7

搜索算法	DFS	BFS	UCS	A*
总时间(s)	0.11	0.18	0.21	0.23
平均节点数	10.17	10.39	9.51	6.85

迷宫大小：21*21

搜索算法	DFS	BFS	UCS	A*
总时间(s)	1.24	8.09	5.45	3.14
平均节点数	114.60	118.23	112.92	49.72

迷宫大小：35*35

搜索算法	DFS	BFS	UCS	A*
总时间(s)	5.64	82.20	44.69	9.78
平均节点数	349.26	344.87	335.22	125.99

从时间角度来说，由于 python 的 stack 与 queue 运行效率差距较大，因此实际结果中深度优先搜索的速度高于其余三种。

从扩展节点数的角度来说，对于大型迷宫来说，A*算法扩展节点数相对于其他三种算法来说要小很多，也从某一方面验证了，当代价函数一致时，A*算法是

最优的。而对于 BFS 与 DFS，在迷宫大小变大的情况下，节点扩展数较大，并不具有很高的空间效率。

六、实验心得与体会

本次大作业工程量较大，不仅复习了不同搜索算法的实现思路，完成了对迷宫问题的建模，还锻炼了代码能力。

由于这是第一次尝试利用 pygame 库进行游戏的编写，而不是用传统的 pyqt，花费的学习时间较长，也感受了真实的游戏制作过程的复杂与艰难,游戏素材图片(字体，音乐)寻找的困难。

此外，在自己观看不同算法的搜索过程中，也对之前略有混淆的概念进行了理解，例如，在第一次小作业中回答错误的下列命题：

如：宽度优先搜索是一种特殊的一致代价搜索

从手动实现的角度对概念解进行了复习，收获颇丰。

七、参考文献

(1)pygame official doc

(2) https://blog.csdn.net/dydlcsdn/article/details/46399111

(3) https://blog.csdn.net/juzihongle1/article/details/73135920