人工智能基础
课程项目 1：连连看

班级：自 92

姓名：魏子卜

学号：2018010734

一、 题目及完成情况

1、 题目要求

在一个 m*n 的棋盘中，散落着 2k 个图案（2k ≤ m*n），这些图案共有 p 类。 相同的两个图案可按照规则进行消除。游戏的目标为，尽可能多地消除棋盘中的图案。

(1) （必做）允许自定义棋盘大小和图案分布情况，在基本消除规则下，设计搜索算法，尽可能多地消除棋盘中的图案，并给出求解过程。

(2) （必做）允许自定义棋盘大小和图案分布情况， 在基本消除规则的基础上，允许转向超过 2 次的连接，转向次数越多，则代价越大。请设计搜索算法，用尽可能少的转向次数来对棋盘中的图案进行消除，并给出求解过程。

(3) （选做）若棋盘中存在若干阻断格子（连接线无法穿过），在与第 2 问相同的条件下，请设计搜索算法，用尽可能少的转向次数来对棋盘中的图案进行消除，并给出求解过程。

2、 完成情况

(1) 必做和选做全部完成。

(2) 可随机产生棋盘，也可以指定棋盘。

(3) 可以指定最多转向次数（默认值为 2），也可以不限制转向次数进行搜索。

(4) 显示转向次数以及消除的过程。

(5) 在棋盘规模较小时实用宽度优先搜索得到最准确的结果，而规模较大时则使用速度更快的算法。

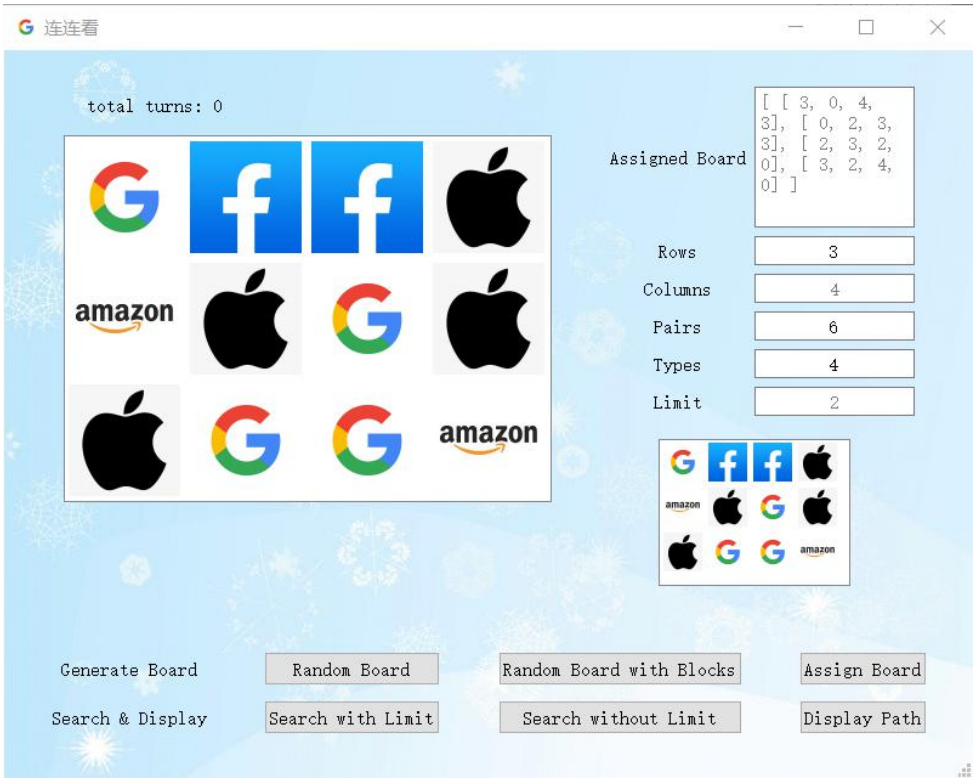
(6) 设计游戏界面。

二、 界面使用方法

1、 概述

界面的主体部分是棋盘，右侧的小棋盘保留了最初始的状态。界面右上角需要用户输入棋盘信息，如果没有输入，则使用预先设好的默认值。界面下方的一组按钮用于产生棋盘，进行搜索并展示求解过程。

需要注意输入应合规，先产生棋盘再点击搜索。如果棋盘中有障碍物，则不能点击”Search with Limit”按钮，原因见后文。在显示”Finish Searching!”前，不要点击”Display Path”按钮。对于较大规模的棋盘（5*5 及以上），其求解速度受图案位置影响很大，如果长时间没有完成搜索，则需关闭程序，重新运行。



2、 产生棋盘(Generate Board)

(1) 随机产生棋盘(Random Board)

依次输入棋盘行数 m、列数 n、图案对数 k、图案类别数 p，则会随机产生一个棋盘，默认值分别为 3、4、6、4。

(2) 随机产生有障碍物的棋盘(Random Board with Blocks)

在随机产生棋盘的基础上，在剩余的位置随机放置障碍物。这里不需要额外的输入信息，完全由随机数决定。

(3) 产生指定棋盘(Assign Board)

在”Assigned Board”栏输入一个二维列表，格式如默认值所示。可以在程序中进行设置，同时程序中预置了多个棋盘。在 processMainWindow.py 的第 xx 行，可以看到程序使用的默认棋盘，并进行修改。

在输入二维列表时，0 代表该位置没有图案，-1 代表该位置是障碍物，从 1 开始增加的整数分别代表各自的图案。需要注意输入的棋盘应符合要求，如图案个数为偶数。

3、 进行搜索并展示求解过程(Search & Display)

(1) 基本消除规则下进行搜索(Search with Limit)

在”Limit”栏输入最大转向次数（默认值为 2），在产生棋盘后，点击”Search with Limit”按钮进行搜索，界面上方显示”Finish Searching!”说明已完成搜索。

考虑到选做题目要求“在与第 2 问相同的条件下”，因此未针对基本消除规则设计有障碍物的搜索。

(2) 允许多次转向进行搜索(Search without limit)

在产生棋盘后，点击”Search without Limit”按钮进行搜索，界面上方显示”Finish Searching!”说明已完成搜索。这里支持有障碍物的不限转向次数的搜索（即选做）。

(3) 展示求解过程(Display Path)

完成搜索后，点击”Display Path”按钮展示求解过程。每点击一次按钮，棋盘显示一次移动或消除的结果，同时屏幕上方的”total turns”显示此时的转向次数，继续点击按钮，直到消除干净或者无法继续消除。此后可以生成新棋盘，同时”total turns”清零。对于较大的棋盘，程序使用了另外的算法，首次点击”Display Path”按钮会将所有直接连接的图案消除（先行后列）。

三、 搜索算法

1、 贪心

贪心法是模仿人类玩家的算法，人们倾向于一上来就将能直接连接的图案消除，使得棋盘剩余图案数量尽可能少。这样解决问题会更快，但也会带来问题，即未得到全局最优解。如图所示的例子，贪心算法进行直接连接的消除，再进行搜索，则总转向次数为 3。而全局最优的总转向次数显然为 2。因此贪心算法会在某些情况下得到不准确的答案。

2、 宽度优先搜索

本项目使用了宽度优先搜索的算法。每一个棋盘会扩展子节点，其子节点包括图案的移动（上下左右移动，不转向）和消除（直接连接）所产生的棋盘。宽度优先搜索的伪代码如图所示。

```
node ← problem.INITIAL(PATH-COST = 0)
IF problem.IS-GOAL(node.STATE) THEN RETURN node
open  ← an FIFO queue with node as the on y e ement
c osed ← an empty set
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF child.STATE is not in open or closed THEN
            IF problem.IS-GOAL(child.STATE) THEN
                return child
            open.PUSH(child)
RETURN failure
```

在实际运行时，较大规模的棋盘可能需要搜索较长时间，因此在速度和准确性上进行了一点取舍。对于 5*5 以下的棋盘均采取宽度优先搜索算法求解，以保证正确性。而对于 5*5 及以上的棋盘，使用了贪心与宽度优先相结合的方法，即初始的棋盘首先会进行一轮直接连接，对于后面的情况，如果能通过直接连接的方式扩展子节点，则不会再通过移动图案来扩展子节点，一定程度上提高了运行速度。

3、 A*算法

有信息搜索能利用问题定义之外的信息，使搜索更具有导向性。但是本算法是从某一个图案进行移动，来扩展子节点，无法预知该图案最终位置，因此无法使用曼哈顿距离来设计评价函数。考虑将当前棋盘的总转弯次数作为路径代价，将棋盘所剩图案数量作为启发函数，但难以设计可采纳的评价函数，因此也没有再使用 A*算法。

四、 代码说明

1、 Block 类

变量：

- row：该图案所在行
- col：该图案所在列
- pattern：该图案，0 代表位置是空的，-1 代表障碍物
- turn：该图案已经转向的次数
- direction：该图案上一次移动的方向，0 代表没有移动过，1 代表向上，2 代表向右，3 代表向下，4 代表向左

2、 Board 类

变量：

- m：该棋盘总行数
- n：该棋盘总列数
- k：该棋盘中图案的对数
- p：该棋盘中图案的种类数
- parent：该棋盘由哪个棋盘扩展而来（初始棋盘的 parent 设为 None）
- spot：该棋盘中正在进行搜索（移动）的图案（[object_row, object_col]）
- spot_init_posi：记录 spot 的初始位置，当无法再扩展子节点时，需要将 spot 还原到初始位置，然后换其他图案再做扩展（[object_init_row, object_init_col]）
- total_turns：该棋盘当前的总转向次数
- block_posi_list：该棋盘的图案，m*n 矩阵，其中元素是 block 的 pattern
- block_mat：该棋盘的图案，m*n 矩阵，其中元素是 block，类似 block_posi_list

方法：

- print_block_mat(self)：以矩阵形式输出棋盘的 block_posi_list，更易于观察。

- generate_child(self)：扩展子节点时，需要产生一个与自身属性相同的棋盘，对不同的部分（某个点的图案，该图案的方向，总转向次数等）再进行修改，同时能实现路径的记录。
- generate_brother(self)：当发生消除后，对剩余的图案都要进行搜索，扩展多个子节点，这些子节点的 spot 是各异的，因此对发生消除后的棋盘使用产生兄弟节点功能，他们的父节点相同，实现了路径的记录。
- available_row_range(self, row, obj_col)：给定棋盘和目标图案（spot）所在行列，返回该图案在一行内可以移动的各个位置。
- available_col_range(self, obj_row, col)：给定棋盘和目标图案（spot）所在行列，返回该图案在一列内可以移动的各个位置。
- direct_connect(self, obj_row, obj_col)：给定棋盘和目标图案（spot）所在行列，返回通过直接连接扩展的子节点。
- available_child_board(self)：给定棋盘和目标图案（spot），返回其扩展的子节点，包含直接连接的消除，以及上下左右的移动
- available_child_board_prior_connect(self)：贪心算法版本，给定棋盘和目标图案（spot），返回其扩展的子节点，如果能够通过直接连接扩展子节点，则不再通过上下左右移动扩展子节点。
- direct_connect_limit(self, obj_row, obj_col, limit)：有转向次数限制版本，在 direct_connect(self)方法上加入转向次数的限制。
- available_child_board_limit(self, limit)：有转向次数限制版本，在 available_child_board(self)方法上加入转向次数的限制。
- available_child_board_limit_prior_connect(self, limit)：有转向次数限制版本 + 贪心算法版本，在 available_child_board_prior_connect(self)方法基础上加入转向次数的限制。
- row_simple_connect(self)：对每一行中能直接连接的两个图案进行消除。
- col_simple_connect(self) ：对每一列中能直接连接的两个图案进行消除。
- simple_connect(self)：先行后列，对图案中能直接连接的两个图案进行消除，即 row_simple_connect(self) + simple_connect(self)，在贪心算法中，对初始的棋盘使用一次，之后不再使用。

3、 search.py

使用宽度优先搜索，同样衍生出贪心算法版本和有转向次数限制版本，实现方法见伪代码，不再赘述。

函数：

- bfs_without_limit(board)
- bfs_without_limit_prior_connect(board)
- bfs_with_limit(board)
- bfs_with_limit_prior_connect(board)

4、 generate_board.py

函数：

- generate_random_board(m, n, k, p)：随机产生棋盘。根据输入的 m、n、

k、p，返回一个随机产生的棋盘。

- generate_random_board_blocks(m, n, k, p)：随机产生有障碍物的棋盘。根据输入的 m、n、k、p，返回一个随机产生的有障碍物的棋盘。
- generate_assign_board(block_posi_list)：产生指定棋盘。根据输入的图案矩阵（二维列表），返回一个相应的棋盘。

5、其他

- main.py：可通过 python main.py 命令运行程序，在界面上操作。
- processMainWindow.py：同 main.py，可通过 python processMainWindow.py 命令运行程序，在界面上操作。
- gameMainWindow.py：包含界面设计的基本信息。
- utils.py：包含队列、优先级队列、栈等数据结构。
- try.py：包含一些测例，以及使用不同函数对测例进行测试，可以简单修改其代码，在终端查看相应输出，以检验算法正确性。

五、 总结

这次大作业是我第一次使用 python 写一个完整的工程，非常锻炼代码能力，同时需要把前面学习的搜索算法用于解决实际问题。对于规模较大的问题，采用贪心+宽度优先搜索的方法提高运算速度。从课上的例子到大作业的题目，都能感受到人工智能学习的知识非常有趣和实用。十分感谢这次大作业的训练，我收获非常大，希望下次大作业能做得更好。