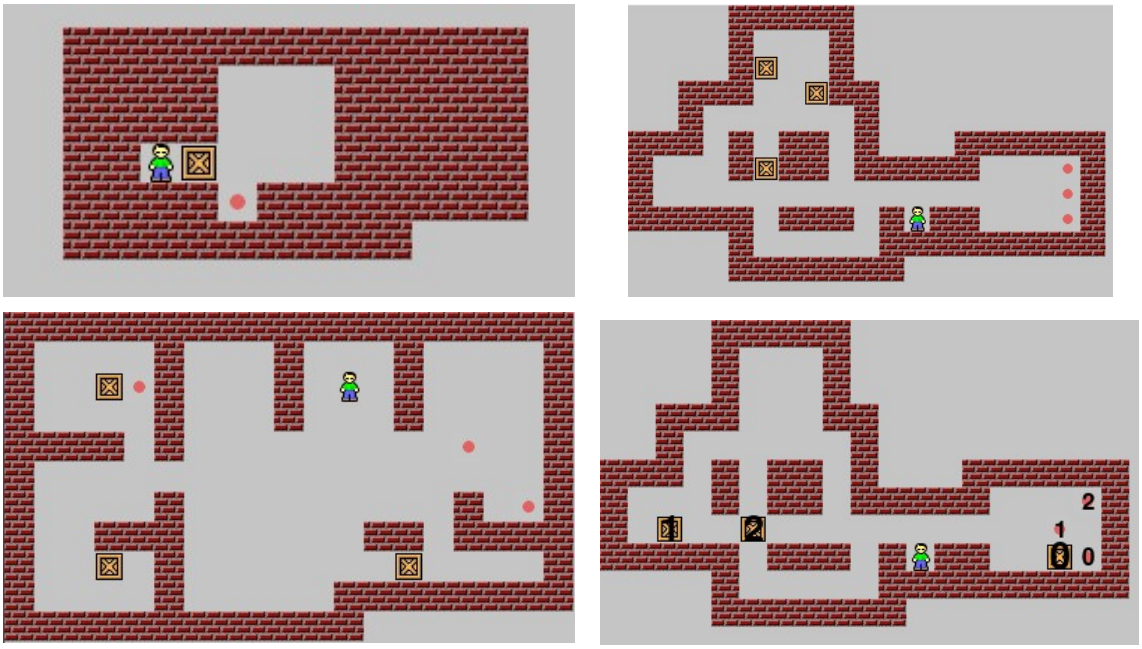
推箱子项目设计

一、 初始场景生成

推箱子的场景需要进行有解性测试，无法简单随机生成，因此预定义了四张地图如下，分别代表简单、中等、困难以及一对一模式。



地图的储存采用了字符串的形式，对于“非一一对应”的情况，使用“@”代表人物，“.”和“+”代表目标，“$”和“*”代表箱子，“#”代表障碍物，“-”代表路面；对于“一一对应”的情况，用“0，1，2”等数字表示箱子编号，“A，B，C”等大写字母表示对应的目标，“a，b，c”等小写字母表示人物位置叠于目标位置上，对于箱子与非对应目标点重合的情况，采用公式 $chr\left(box_{idx}+10\,target_{idx}+ord(A)\right)$ 将箱子与目标的编号重新编码并转为字母储存（该方式断言箱子数小于 5）。

二、 UI 使用方法

对于界面 UI 的使用方法，在开始界面选择地图，即单击所选难度字串中心，之后可观看搜索算法对该图求解得到的最优路径图形化展示。由于 UI 用于展示搜索算法，故未提供手动游玩模式，但在自动求解结束后可以使用键盘“上下左右”键进行一些额外的手动的操作。

大量的搜索过程渲染回占用大量时间，同时也会使得 pygame 卡顿，故 UI 仅展示搜索得到的最优路径而不展示搜索过程。同时为节省搜素时间，提升用户体验，我储存了搜索算法给出的所有地图的最优路径，在渲染过程中直接调用。若要验证搜索算法的正确性，请将源代码第 552 行代码去注释并删去第 554 行后，直接运行源代码（非 exe 文件），地图较大，请耐心等待较长时间。

三、 非一对一情况求解

对于此部分，我以地图和人物位置的元组作为状态，设计了广度优先搜索和 A*搜索算法，根据两种算法的完备性和最优性，若地图有解，则一定找到最优解。值得注意的是，两种算法均是在不同的状态中进行搜索规划，而非传统路径规划中对人物前进路线进行搜索规划。

由于 A*算法性能优于广度优先搜索，故在此着重介绍 A*算法。我以所有箱子距离其最近的目标点的曼哈顿距离之和为启发函数，鼓励算法向着箱子更靠近目标点的方向搜索。在搜索的过程中，自定义了地图更新规则，每次迭代均更新地图，同时标记了已搜索过的地图状态，进行剪枝操作。

我将 BFS 与 A*的搜索路径进行对比，发现二者最优路径步数均相同，因此认为算法最优性正常。同时二者搜索到的路径有所差异，这应是两种算法内核差异导致的。

四、 一对一情况求解

对于此部分，我对状态的表示与非一对一相同，由于非一对一情况使用了 A*算法，故在此设计广度优先搜索算法，根据算法的完备性和最优性，若地图有解，则一定找到最优解。在搜索的过程中，我自定义了地图更新规则，每次迭代均更新地图，同时标记了已搜索过的地图状态，进行了剪枝操作。

事实上，在我的搜索思路下，一对一情况的搜索与非一对一情况的搜索除了地图更新策略以及终止条件不同外，其余部分可以保持一致。这在侧面证明了我算法思路的稳定性和鲁棒性。

五、 反思

本项目中，我学习了 pygame 前端的搭建，加深了对 BFS 和 A*算法的理解，第一次真正在非“传统寻路”的层面应用图搜索算法。在搭建前端和编写算法的过程中，我遇到了许多不可预期的问题，包括 UI 界面更新异常以及部分地图求解异常，但是在查阅资料和不懈努力最终得以解决。在以后的学习实验中，我会勤加练习，注重理解，同时也会在以后的代码编写过程中更加谨慎，以保证算法正确稳定。谢谢助教老师耐心阅读，祝您游玩本项目顺利愉快！