课程项目一

这次课程项目我的选题是连连看，下面从三个部分描述一下。

1.问题建模

首先定义问题的状态空间。将连连看棋盘的每个排列情况看作一个状态，其中初始状态就是开始时棋子的分布，目标状态就是每个格子都没有棋子，即完全消除的情况。每一步的行动则是一次消除，即从棋盘中选择两个合法的棋子进行消除。由于本次问题中对转向次数做出了限制，因此这里将转向次数定义为行动的代价（或者和入队的顺序相关）。

综上，问题的状态集就是所有棋子（包括空格）可能的排列，行动集就是在当前状态下所有合法消除的集合，代价函数就是转向的次数。

除了整体的搜索算法外，在寻找每个棋子的合法配对消除棋子时，也使用了搜索算法。在这个子问题中，状态空间定义为当前节点的坐标和棋子类型（包括空格和石头），其中起始状态为待搜索棋子类型和起始坐标，目标状态仍然是该类型和一个不同于起始的坐标。动作集合包括了上下左右四个方向的移动，而代价函数也是转向的次数。

2.搜索算法

介绍了问题的建模后，具体介绍搜索算法。首先外层的搜索（以全消除为目标对棋盘状态的搜索）使用了一致代价搜索，这样可以找到代价最小的解。

首先在第一个问题中，由于没有限制转向次数，因此使用 DFS 求解即可，可以加快求解速度。为了将一致代价搜索退化为 DFS，我们先对初始节点定义了很大的代价，然后每次扩展的子节点代价为父节点的代价-1，这样保证了后入队的元素有更高的优先级，即实现了 FILO 的结构。

然后讨论子节点的扩展。由于动作集是寻找合法的消除对，因此这里使用了一个内层的搜索对每一个棋子进行检查，得到所有可能的消除对，即外层搜索的子节点。对于转向超过次数的消除对，我们将其定义为非法的，即把问题交给内层的搜索解决。这样外层搜索的问题就基本解决了。

在第二、三问中，只要正常执行一致代价搜索即可。第二问中的代价由父节点代价加上动作代价得到子节点代价，其中动作代价由内层搜索提供。第三问中的石块问题，我们可以将通过石块的路径定义为非法动作，进而把问题抛给内层搜索。

对于内层搜索，由于第一问有对转向次数的限制，我这里仍然使用了一致代价搜索。因为我这里对 depth 的定义仍然是移动的步数，所以有可能出现这样一种情况：算法先沿着一个转角较多的路径到达目标，但是由于代价过大而被否定，同时目标被加入 closed 表。这样当算法再次沿着一个步数较多但是转弯较少的路径（合法）来到这个状态时，就直接被 closed 表否定了，因此错过了一个可能的解。如果将转弯数定义为 depth 并使用 BFS 可以解决这个问题，这里我将转弯数当作了 cost，因此需要使用一致代价搜索找到最优的解。另外由于内层搜索在第二、三个问题中需要为外层搜索提供动作代价，因此需要为每个消除对找到最小代价解，因此仍然需要一致代价搜索。

在三个问题中内层搜索的逻辑基本类似。在第一问中，需要对代价大于 2 的节点进行剪枝，以满足题目要求。第二问中不需要进行这一步剪枝，但是仍然要求出消除对最小的代价，提供给外层循环。对于第三问，当路径搜索到石块时进行剪枝即可。

综上，对每个问题的整体解决思路如下：

当游戏开始时，对整个棋盘执行外层搜索。如果是第一个问题，则执行的是 DFS，如果是后两个问题则执行的是一致代价搜索。在扩展子节点时，对每个棋子进行检查，找到整个棋盘上所有可能的消除对。

检查具体使用了一致代价搜索，为每个棋子找到所有代价最小的消除对。在第一问里需要对代价大于 2 的节点进行剪枝，而在第三问时需要对遇到石块的节点进行剪枝。

搜索完毕后将路径存储在 solution 里面，每次点击 hint 按钮时弹出一个步骤。

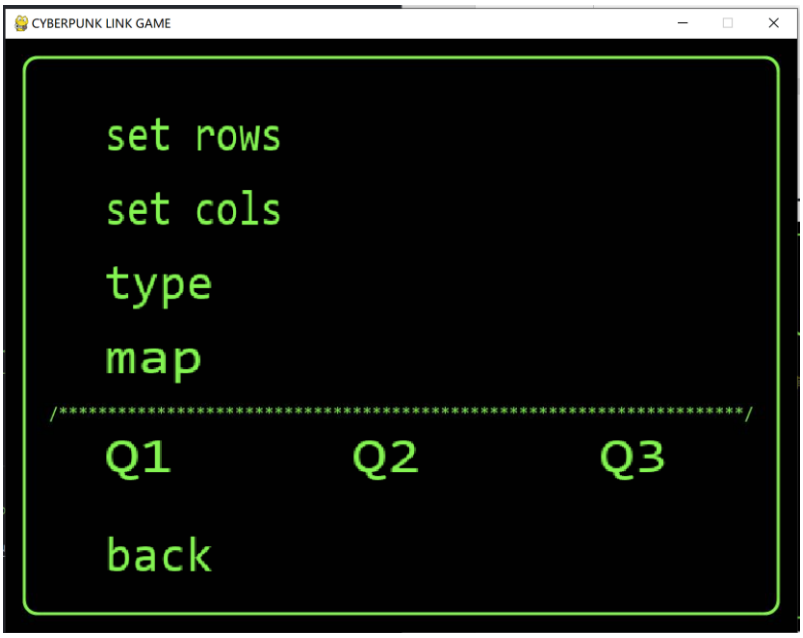
如果想人进行游戏，直接点击即可。每次选中一个棋子时，对其执行上述内层的搜索，找到所有可能的解，同时仍遵循上面的剪枝规则。点击第二个棋子时则检查其是否在解中，如果在则执行消除。由于消除后整体解的路径可能发生变化，因此再次执行外层搜索，找到当前的整体解的路径。

3. 程序流程

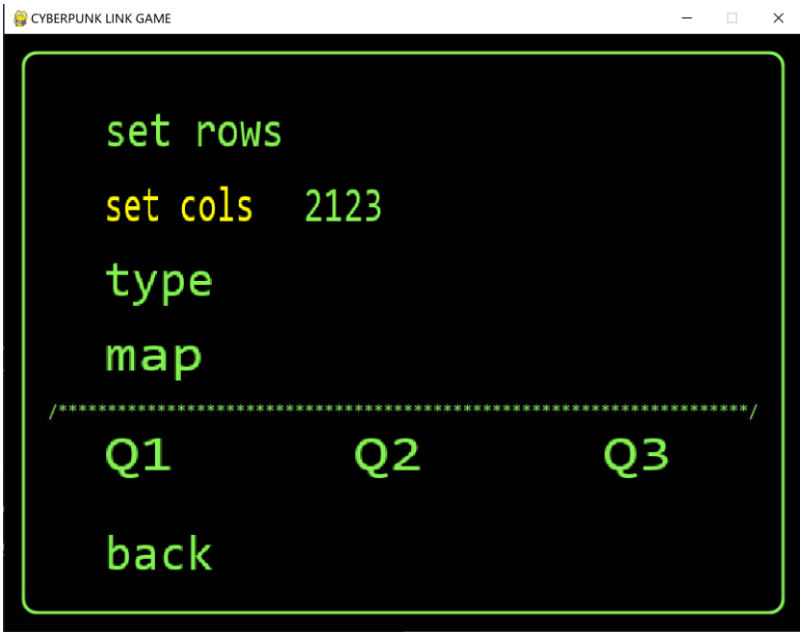
首先程序启动时，加载的是启动界面。

下方两个是可点击的按钮，分别表示开始游戏和退出。鼠标放在按钮上会产生提示效果。

点击开始游戏按钮后进入设置界面。上半部分是用户自定义输入部分，可以在这里定义棋盘大小，棋子种类和具体地图。输入仅限于数字和‘x’字符。‘x’表示地图中的石块。

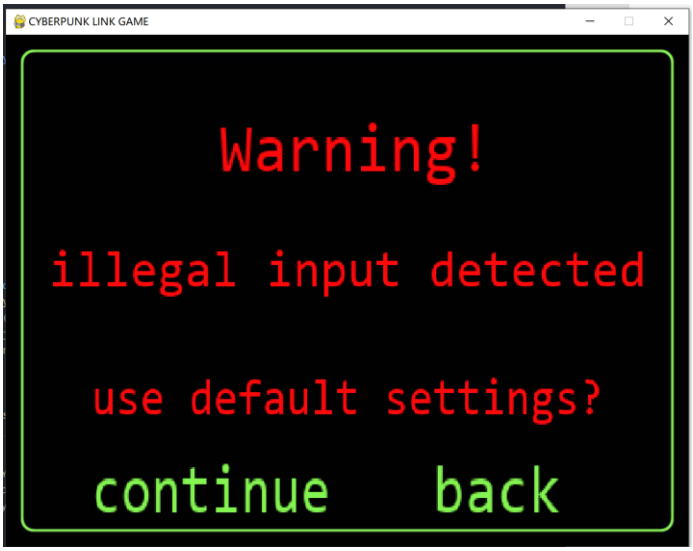


如果要输入某一行的内容，则点击该行的标题即可。标题会产生提示效果并展示输入的内容，按退格可以进行修改。只有高亮状态的栏目才会产生有效输入。



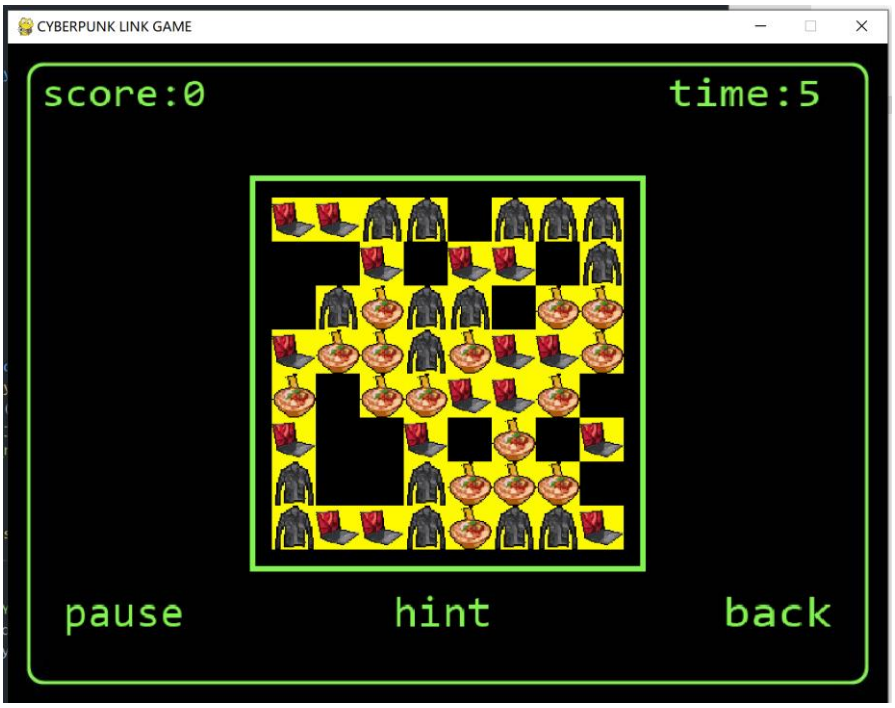
下方是按钮区，分别表示启动三种问题的规则。还有一个返回主菜单的返回按钮。

如果输入不合法，例如棋盘尺寸是奇数，或者缺少某些项目时，会进入警告页面，如下图：



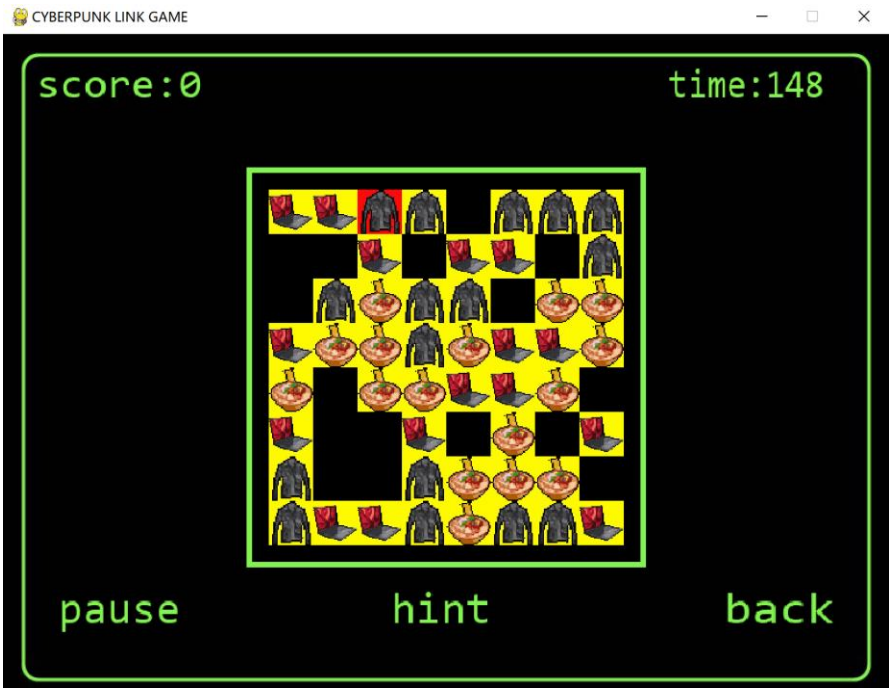
可以选择使用系统默认的设定（点击 continue）或者返回主菜单重新设置。

进入游戏界面后，整体如下图：

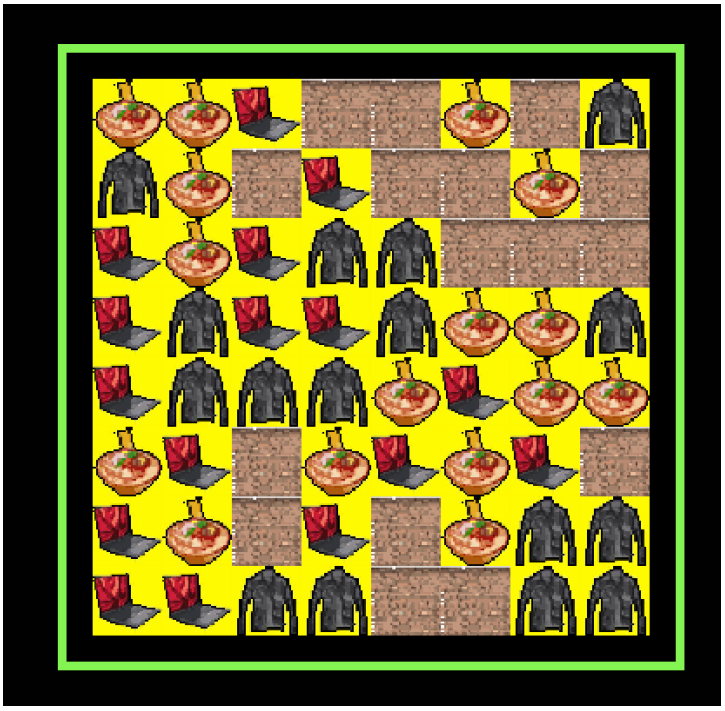


中间是棋盘，左上角是分数，右上角是时间（秒为单位）。每次手动消除一个合法消除对后分数会+10，另外时间会记录本次的游戏时间。

中间的棋盘有不同的棋子，图案一样的棋子可能可以消除。未选中的棋子是黄色背景，而被选中的棋子会变成红色背景，如下图：

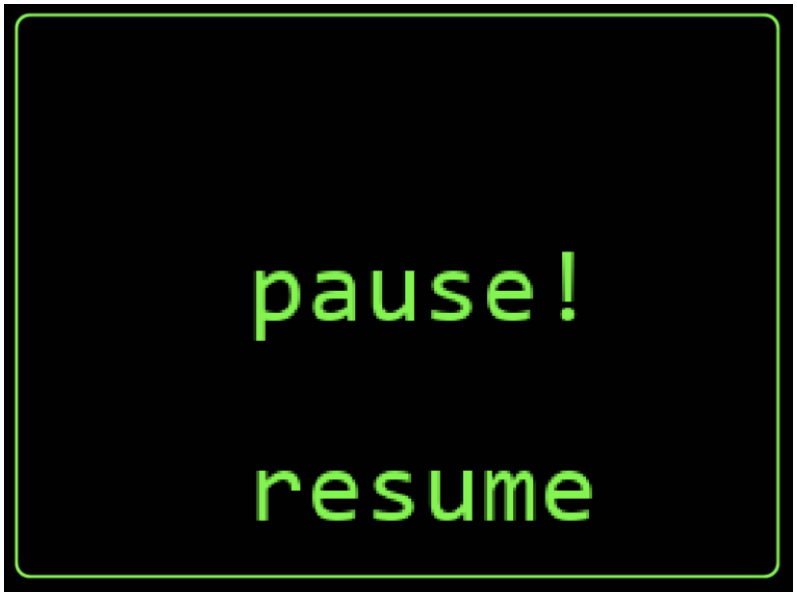


表示该棋子被选中，如果再点击一个合法格子，这两个棋子就会消除变成黑色，黑色的格子表示空的位置。如果出现下方所示的棕色方块，则表示是石块，不可选中和消除。



下方的三个按钮分别表示暂停、提示和返回。

点击暂停按钮，计时器会停止计时，进入暂停页面，点击恢复即可恢复刚才的残局。



点击提示按钮则会自动消除一个合法的消除对（按照题目要求的整体路径），连续点击提示按钮即可得到求解过程。点击返回按钮可以返回主菜单。

当所有棋子消除后，游戏结束，进入结算页面，显示分数和返回按钮。点击返回按钮可以返回主菜单重新开始。

4. 不足和注意事项

首先这个程序最大的不足就是速度过慢。其主要的原因是对外层执行了一致代价搜索。由于代价定义的是转角次数，因此这个代价增长比较均匀。这意味着我们几乎要遍历所有的状态才可能找到一个最优解。如果想解决这个问题，可以使用 A*算法进行有信息搜索，但

是我没有想到比较好的启发函数，因此没有采用这种方法。如果有一个比较好的启发函数，可能效率会得到很大的提高。

另外一些细节上我也没有注意效率，例如对 PQ 的实现，我只在逻辑上完成了 PQ 的功能，没有使用堆等结构进行优化。还有一些细节上我用了很多循环，这些都有优化的空间。

最后在使用该软件时有一些注意事项。首先由于界面大小限制，棋盘大小不要超过$8x9$。对于问题一，$8x9$大小的棋盘可以完成计算，但是每次点击后需要等待$2\sim3$秒进行计算，连续点击容易造成卡死。对于问题二三，我尝试的可以较快得到解的最大尺寸为$4x5$。这种情况下进入后需要等待 10 秒左右进行求解。此时不要点击程序，否则容易造成卡死（当计时器正常走动时即可点击）。而大于这个尺寸的棋盘可能无法较快求解。

最重要的是，由于 while 循环中每次刷新需要一定时间，因此最好不要快速点击或者连续点击，否则可能会在一个 while 中判定两次事件，造成未定义的行为。有的时候程序（尤其是发布的可执行文件 exe）会出现莫名其妙的缩进报错，再次尝试启动即可解决。