课程设计报告

第一阶段（Phase1):

1、基础工作（选择合适的GUI库）：

选择了Pygame库，找到了对应的教程并开始学习。
Pygame库提供了易于使用的鼠标控制、画面处理、简易动画制作等功能，因此是制作这个类型和规模的程序的最佳选择。

学习了Pygame的使用，并通过一些小的前置练习明白了如何处理点击、鼠标移动、键盘敲击，也学习了如何处理图片、发送定时信号、绘直线、矩形、圆弧等。

2、界面制作:
首先决定了游戏界面的排版，采取了左侧为游戏内容（主要画面）、右侧为按钮的布局。

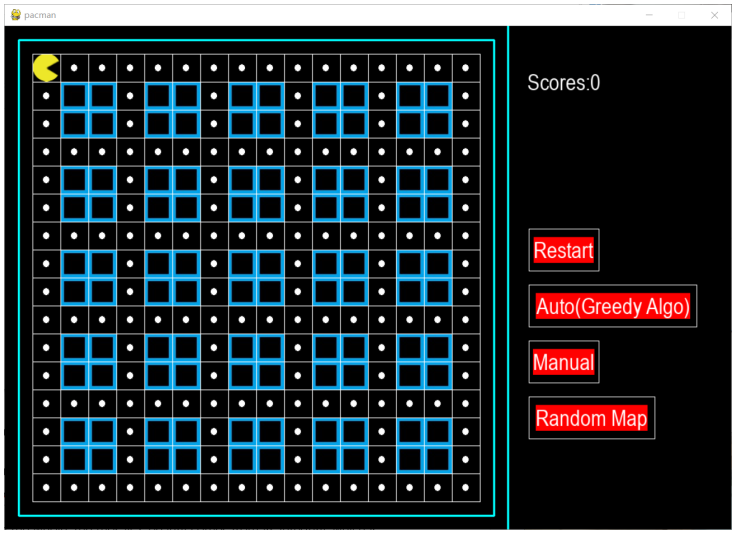
学习了本来吃豆人的地图排版，意识到地图可以被分割为格子，而吃豆人的地图大体上是网格状的离散布局。

按照这个理念，也为了简化地图的设计，决定用矩阵来生成地图。大体上，决定采用0代表空地，1代表墙壁。

思考地图的大小，决定采用16*16的设计，每个格子为40*40pixels，这样可以让画面总体出彩。
于是绘制了不同方向的吃豆人，从网上选择了合适的“墙壁”图片，设计出了最初的地图，同时绘制了网格（后来已去掉）。

在这个阶段采取的是“离散地图”，即吃豆人不是自动移动的，而是只有敲击键盘后才移动，类似于“棋类”游戏。同时，这时的“按钮”只是一个“事实上”的功能，就是敲击鼠标时检测其落点，如果落在给定的“区域”之中，则执行对应的函数。

随后加上了豆子和“吃豆子”的函数，最基本的游戏设计就制作完了。



3、功能设计：

剩下的第一阶段部分是“功能”而不是“图像”，因此我想依次介绍我实现的功能，用文字而非代码。

首先是吃豆人的动画，在原版游戏中，吃豆人的嘴是“一张一合”的，于是我使用了一个定时的信号，定期刷新吃豆人的形态，即“张嘴”或“闭嘴”。

随后是地图生成。我选择结合了两种生成方式，即“游戏内生成”和“输入生成”。由于地图只是一个矩阵，因此地图的生成并不难。

再然后是随机地图，这是一个看上去不难，但实际上十分复杂的功能。我设计了一个参数，这个参数在总体上将控制地图的疏密（我的设置是这个参数是位于0-1之间的小数，数越小地图中墙越少）。

随机生成算法如下：
首先假设地图中无墙，随后依次遍历每个格子，首先看这个格子是不是关键格子（就是如果这个格子变成墙会不会让地图不连通），如果不是按概率把这个格子变成墙。

4、“连续”的吃豆人游戏：
第一阶段的最后一个部分是把吃豆人变成“连续”的游戏，但如何使吃豆人连续的跑动？
第一个想法就是让地图快速刷新，每次刷新时自动“更新”吃豆人的“实际位置”。但这样会让吃豆人转向变的极为困难，因为只有一瞬间吃豆人位于可以转向的位置，其他时候允许转弯必然会使得吃豆人和墙“穿模”。
如何使操作更为流畅困扰了我挺久。向同学请教经验时，他给我的“将实际坐标和动画坐标分开”的建议让我知道了如何改进。我因此设计了“实际坐标”和“动画坐标”，“实际坐标”是瞬间移动的，而“动画坐标”永远在从之前坐标走到实际坐标中，因此就有了动画效果，游戏变的“连续”了。

第一阶段-改（Phase1-Update）：

在第一阶段做完之后，做第二阶段之前，我对第一阶段的内容进行了较大的修改。应该说，整个第二阶段中我有约一半的时间是在进行第一阶段的更新，因此我将称之为“第一阶段改”。

1、重构代码：
第一阶段做完时，虽然功能均已经实现且没有bug，但所有代码都写在一个文件中，而且变量、变量作用范围均十分凌乱，于是我开始了漫长的重构。

本学期的课程中我学习了“面向对象程序设计风格”，因此我决定重构代码，将代码的架构调整。
因此我将程序拆分为了多个文件，每个文件各司其职。目前的作用是这样的（省略后缀py）：
MainLoad负责主运行，Control负责自动吃豆、鬼的追踪和吃豆人的智能（将在第二阶段、第三阶段投入使用），Agents定义了Pacman和Ghost的行为，Painting负责刷新画面，SrcLoad负责初始化游戏和加载图片的功能，Parameter中定义了一些重要的参数、Util中定义了Maps、Button、Directions这三个辅助类和对应的辅助函数。
我将本来只用零散的函数和变量定义的吃豆人、鬼、地图、按钮封装为了类，此中我感受到了“封装”的作用。比如说，设计吃豆人的行为的控制函数本来分散在各处，现在我可以集中至一个类；本来按钮的悬停、动画设计、位置需要一个按钮一个按钮分别控制，现在只要一行语句就可以创建一个新按钮。
同时，每个文件负责各自的功能，让代码更为清晰了，对应的功能（比如显示或者智能）可以分开编写。

2、改bug&小优化：
我处理掉了第一阶段中遗留的bug，也优化了某些小的方面，但总体上我并没有对功能上进行较大的优化。
以下是写于“readme”中的bug修改日志（划掉），不过其中这样“优化”让我对大作业有了更浓厚的兴趣。

有所改进的是：
取消了绝对路径，修改为了相对路径
修复了运动时可以转向到墙壁方向的设计缺陷
略微重置了UI，使得悬停在按钮上和按下时有了动画
改进了吃豆人的动画，现在它的嘴动起来更好看了
略微修改地图样貌，使得地图更好看了一点
增加了全部吃完后提示的"You win!"

第二阶段（Phase2)：

1、代码模块的功能划分：

1. Agents.py中定义了两个类，Pacman类和Ghost类，每个类中详细的定义了对象可以做的行为，如吃豆人会吃豆子、吃胶囊变成无敌状态、移动、转向等；而鬼则会追击、逃跑、像原版一样，每个鬼还有自己的行动特色。Pacman类有许多成员，其中有好几套坐标，这是为了和“连续动画”相一致；Ghost类也是。
2. Painting.py中定义了repainting函数，这是GUI的核心函数，这个函数保证了游戏的画面刷新；以及辅助函数paintingbuttonborder，负责画按钮的边线。
3. SrcLoad.py中定义了开始游戏会先加载的函数，分为input_file（从map.txt中提取默认地图），imageinit（提取图片），fontinit（加载字体）、initset（设置窗口等）、agentinit（初始化地图和对象）、setbutton（设置按钮）。
4. Util.py中定义了三个类：Map类、Directions类、Buttons类。Map类定义了地图的各个行为（比如重置地图、初始化地图、查询地图中每一格的值、查询地图的食物数量等）；Directions类是作为接口，规范化吃豆人和鬼的行为；Buttons类封装了按钮（显示按钮、点击按钮），使得可以“一行”创建一个新按钮，如Qt-C++和VB一般。
5. Parameter.py中定义了许多通用的参数，其中绝大部分不应该修改，不过仍然有两个可以修改的参数（比如说设定地图中墙壁密度的参数）。
6. MainLoad.py是地图的主文件，主要控制事件的while循环。游戏事件分为几种，如鼠标敲击、键盘敲击、鼠标移动、以及自定义信号，MainLoad中的main函数其实就是处理这些事件的。
7. Control.py是控制寻路的智能算法，具体将在下一部分进行阐述。

2.设计算法：
我设计了四个算法，主要是基于搜索、启发式搜索和动态规划算法。同时，在四个“成功”的算法之外，也遇到了几个遇到了反例的失败“寻路”算法。

在第二阶段，我主要学习了各种搜索算法，在CS188课程中也学习了入门的强化学习方法。不过第二阶段中我暂时没有写强化学习算法，而是打算在第三阶段中正式进行实现。
在叙述算法之前，我打算先阐述一下第二阶段的任务及其“变种”，从而更好的为接下来阐述算法做铺垫。
第二阶段的“基本”任务主要是：考虑只有豆子和吃豆人地图，设计将豆子可以尽快全部吃完的算法。
第二阶段的“变种”任务是：豆子很少（只有10颗左右）和只有一颗豆子的情况。
后两个任务是为前一个任务做铺垫的。
只有一颗豆子的情况和豆子很少的情况，是可以找到“绝对”的最优解的；但当豆子很多时，就只能选择“较优解”。但仍然有些思想是通用的。

Dynamic Programming（动态规划）：
先说DP算法，因为DP算法是一个确定的算法。DP算法基本只能在豆子小于15颗（如果性能优化后可以到20颗）时使用，用一个2进制数来表示吃掉某些豆子的算法。
比如假如有5颗豆子，状态就是五位二进制数。假设就取名为DP数组（个人习惯），DP[01100]即吃掉第三颗和第四颗豆子的最小代价，DP[10001]即吃掉第一颗和第五颗豆子的最小代价，自然DP[00000]=0，因为没吃掉豆子的代价自然为0。
现在根据吃豆子所在的位置，算出吃豆人走到某个豆子的最短路径dis，进行更新。
比如现在状态是DP[01100]，吃豆人位置在第三颗豆处，算出吃豆人到第一颗豆子的距离D后，就可以进行状态更新，DP[01101]=min{DP[01101],DP[01100]+D}。
当然DP数组是个二维数组，第二维数组的大小为2，分别存“最小距离”和对应的“位置”（但此处暂且省略第二个）。根据此就可以一步步更新到DP[11111]，值即为所有可能的路径中最小的。
因此，可知DP算法的空间复杂度为$O(n*2^n)$，经计算得时间复杂度为$(n2*2n)$，在n过大的时候使用DP算法是不现实的，因此我设定为当豆子过多时DP算法不会执行。
DP算法只在”OneFood Map”和”Scarce”地图中有效。

搜索算法：
普通搜索算法主要是两种：DFS（深度优先搜索）和BFS（广度优先搜索）。
当豆子较少时，DFS/BFS也可以通过搜完所有的豆子来寻找最优解（遍历），但对于较多豆子时就不能如此做。因此为了解的通用性，剩下三种算法没选择全部搜完的方式，而是选择了另一种方式。
具体而言，在取舍之后，选择了这样的思路：设定某种代价，以最小化代价的前提找到“最优的下一颗豆子”，吃掉它，然后再做选择。
三种搜索算法都是如此，但是DFS/BFS和启发式搜索中设定的“代价”不一样。
DFS和BFS中的目标比较简单，就是“第一颗找到的豆子”，因此算法可以简化为“找到第一颗没吃掉豆子，吃掉它，然后继续找，直到吃完为止”，这两种搜索的区别在于找到最近豆子的方法，是深搜，还是

广搜。经检验，绝大部分情况下广搜优于深搜。（豆子极少时例外）
因为广搜保证找到的一定是最近的豆子，而深搜有可能找到相当远的一颗豆子（但在搜索过程中，它确实第一颗被找到的）

启发式搜索：
启发式搜索中的代价函数是“所有豆子”到某个点的曼哈顿距离之和，因此吃的下一颗豆子不一定是最近的，但吃掉这颗豆子后可以让位置和剩下所有位置之和较近。
因此我设计这种算法，希望在另一种程度上能让吃豆人有一定的“大局观”。

启发式搜索之前：
关于这个启发式搜索，是存在一个废案的，就是按照“所有豆子到某个点的曼哈顿距离之和”来决定每一次怎么走（而不是下一个吃什么豆子），但这样的算法存在死胡同，考虑如下情况。

这种情况下，黄色位置的吃豆人到三颗豆子的总曼哈顿距离是6，但吃豆人一旦走了一格（无论哪个方向），总曼哈顿距离就变成了7，也因此吃豆人会“徘徊”，它没法跳出这样的“局部坑”。
因此这个算法是失败的。

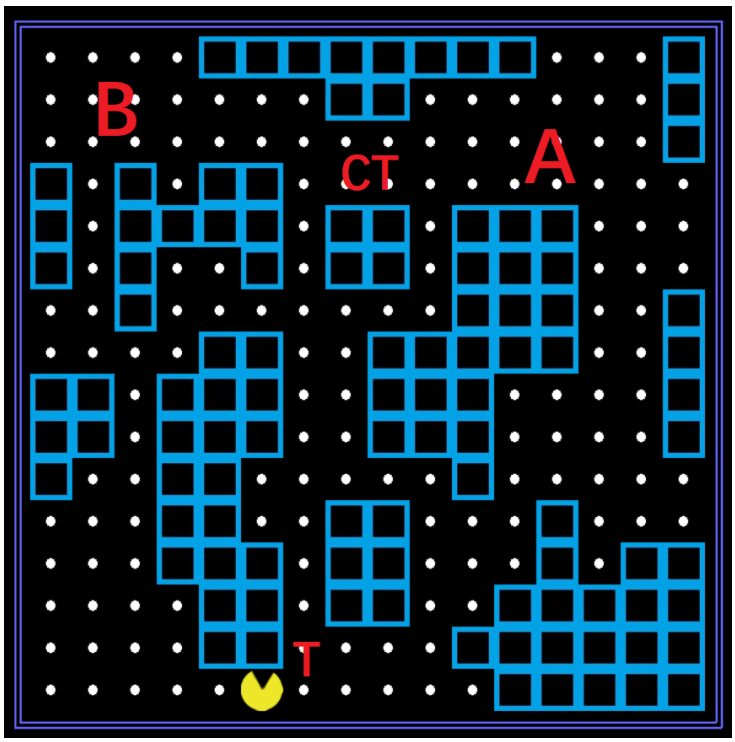
可能的改进1：
由于墙的存在，所以曼哈顿距离并不一定是实际距离，任意两点的实际距离都可以在开始通过Floyd算法算出，或许这是可以改进的一个点，但当豆子较多时，Floyd的O(n^3)复杂度将影响游戏运行。

可能的改进2：
对于这种不必求“最优解”的问题，或许一个改进是采取比较著名的启发式算法，如“模拟退火算法"等，但由于这个阶段我在学习强化学习算法，所以就没来得及写这种算法。

地图设计：
为了更好的设计并且对算法进行细节上的优化，我设计了如下几种地图。
Default：基本地图，但未采取吃豆人游戏中的地图，而是设计了网格地图。
Random：随机地图（一定连通），地图中墙的数量可以通过参数控制。
OneFood：随机地图，只有一个食物

Default Scarce：基本地图，只有零碎的食物
Random Scarce：随机地图，只有零碎的食物
~~Dust2：看CSGO比赛上头的产物，模拟了Dust2的地图，作为彩蛋，并没有在Phase2的阶段中将这个地图丢进按钮中~~

（大雾）



算法效率统计：
首先来看一张表格（表格内数字是吃完所有豆子所需的步数，随机地图/稀少食物自然受随机性所影响，因此除了确定的地图外，都只有统计结果）

Algorithm	Dynamic Programming	BFS	DFS	Heuristic
Default Scarce	About 57	About 63	About 80	About 70
Random Scarce	About 62	About 66	About 85	About 72
OneFood Map	Optimal	Optimal	Optimal	Optimal
Default Map	/	200(Exact)	337(Exact)	214(Exact)
Random Map	/	270-300	500-1500	350-450
(Bonus)Dust2 Map	/	192(Exact)	658(Exact)	214(Exact)

这是不同算法在不同地图中的表现，基本为DP<BFS<=Heuristic<<DFS（当然，DP必然是最优解，但DP自然有其限制）
DFS随机性影响很大而且普遍性能最弱，因为DFS找到下一个豆子是很大程度上“不确定的”，而BFS略优于Heuristic说明当前设计的启发函数还不够优秀。
在稀少豆子的地图中，各种算法的差距不大，但豆子更多的时候，差距就变的大了起来。由于此时想找到最优解几乎不可能，只能想办法让AI有更优的“大局观”和“智能”

第二阶段总结

第二阶段的重点在于拆分问题，并且实现“搜索并尽快的吃掉所有的豆子”。
因此，在学习了第二阶段的参考资料"Project-CS188"之后，我将问题拆分成两个变种，在实现这两个变种的同时渐渐地实现原本的问题。
第一个变种是“只有一个豆子”，这时可以通过搜索（启发式搜索）、单源最短路径、动态规划来解决这个问题。
第二个变种是“有数个的豆子”，此时可以通过搜索（启发式搜索）、动态规划来找到最优解，也可以用“一个豆子一个豆子”的搜索来找到较优解。
随后是原本的问题“有许多的豆子”，此时想找到最优解已经不太现实（接近旅行商问题），因此主要是通过贪心搜索和启发式搜索找到较优解。

在这个阶段中，我一方面是优化了程序的架构，使得程序可以较好地兼容各种不同的搜索算法（有的算法是离线的，在最初时就可以全部算好；有的算法是在线的，是一步步考虑的，要兼容这两种算法需要较好的程序架构）。
另一方面，在这个阶段我在试图采用各种不同的非人工智能算法来实现“搜索的智能”，同时也在学习CS188课程和强化学习算法。不同的搜索算法确实有不同的表现，我在程序设计的过程中深有体会。

第二阶段的设计内容就到这里了，我希望在第三阶段可以圆满的完成这个游戏、这个大作业，也可以入门人工智能算法和各类算法。
除了作业本身学到的“智能”知识之外，这样一个“设计游戏”（逐渐创建新功能、运行、调试bug）的过程对我程序设计水平也是大有帮助的。
第三阶段再见(*^▽^*)~

第三阶段(Phase 3)

游戏逻辑和游戏模块设计
第三阶段的第一部分是设计-出胶囊和鬼。因此，我首先陈述这两个部分。

I 胶囊

首先是关于胶囊。由于我设计的Map类统一了地图、墙和食物，因此，胶囊只需要被视为特殊的食物即可。

不过在此产生了一个问题。当Pacman吃掉食物时，它必须获取Map中的元素，因此Pacman类是引用Map类的。如果在Map类中添加函数来使得吃掉豆子后的吃豆人无敌，则必须修改Pacman的状态，这样的话Map类就会引用Pacman类。两个类分别位于两个文件中，而如果两个类互相引用会导致程序报错（循环引用）

对于这个问题，我选择在主程序中添加代码来记录吃掉豆子后“变成无敌”的功能，这样就避免了循环引用。

II 鬼

然后是关于鬼。鬼的诸多特性和吃豆人是一样的，比如鬼也有位于不同方向走的时候的贴图，有“连续移动”的特性，可以像吃豆人吃掉豆子一样吃掉吃豆人。而且，不同于玩家操控的吃豆人，鬼需要有它的寻路算法来自动找到吃豆人，从而达到“追击”的效果。

鬼有若干种算法，不同的算法可以实现不同的鬼。我这里设计了若干种不同算法的鬼，它们贴近了原版的鬼，让游戏难度更加高了。

首先是随机鬼。随机鬼是随机游荡的，智能性最弱。

然后是最优化鬼，沿着最短路径沿着吃豆人走，这样的鬼是最强的。

考虑到最优化鬼后可以设计部分最优鬼，即较大概率走“最优化”路线，否则走随机路线，仿照强化学习中的“epsilon-greedy"算法的命名，我称其为"epsilon-ghost"。

然后有最优化鬼的变种，堵截鬼，就是以吃豆人的前方为目标，试图“堵截”鬼。

我的设计也部分取自于原本的吃豆人游戏。

在原版的吃豆人中，四个鬼有不同的名称和思路：

红色的Blinky，“执着”，会一直跟着玩家走。（最优化鬼）

粉色的Pinky，“预知”，会围堵在玩家前进的路上，埋伏袭击。（堵截鬼）

青色的Inky，“变化”，有时会追着玩家走，还有的时候会堵在前进路上，变化不定。（前两种的混合）

黄色的Clyde，“随意”，行动路线完全随机，并不理会玩家，但在被其他幽灵围堵的情况下，难免不碰上。（随机鬼）

当然，鬼还需要有恐惧后“逃跑”的智能，这时我就采取了和之前相反的代码，即鬼会朝着离吃豆人最远的方向跑，同时有一定概率随机奔跑（否则几乎无法被吃掉）

III 交互

制作完鬼和胶囊后，重要的就来到了如果定义交互，交互并不是一个很复杂的事情，只要判断鬼和吃豆人的距离、鬼是否出于恐惧状态即可。

在修改了不知道多少个bug之后，“系统”就设计完毕了。

1-IV 新模块

我新建立了AgentControl.py，在其中设计了各类智能算法。具体的内容我会在接下来陈述。此外，我还

建立了Qtable.py模块，这个模块会在之后给出解释。

智能算法

第三阶段中，我设计了四种智能算法，两种是搜索算法，分别是基于规则的躲避-逃跑算法和Minimax搜索（以及Alpha-beta剪枝优化）；另外两种是强化学习的基本方法，即Q-学习算法（根据参数的不同，分别趋向于蒙特卡洛采样和时序差分算法），以及Sarsa算法。每种算法都有其特点，但也有其局限。和第二阶段类似的，我也简化了问题，分别设计了两种子问题和一个带有局限性的完整问题。有些算法不一定可以解决完整问题，但可以解决子问题。

Sub1.地图中只有吃豆人和鬼，此问题中吃豆人不必考虑如何尽快的吃掉豆子，而是尽可能在鬼的追击中生存更长时间。
Sub2.地图中有鬼、吃豆人和豆子，但是没有胶囊，这时可以在不考虑“鬼的恐惧/无敌吃豆人”的另一套逻辑时，吃豆人如何平衡鬼的追击和对豆子的获取。
Sub3.地图是完整地图，具有鬼、吃豆人、豆子、胶囊，但是地图是较小的。

规则算法

第一个算法是我写的规则算法，即将一些知识通过代码（比如说许多的if-else）来教会吃豆人去学会吃豆子。

我设计的第一种规则算法被我称为"Avoidance"，因为这个算法没有考虑胶囊的存在，而是通过简单的情况判断来是否应该吃豆，或者躲避鬼。（具体见AgentControl.py的avoidance_pacman()函数。
第一种规则算法的原则十分简单：如果和最近的鬼的距离小于6，就尽可能跑；否则就奔向最近的豆子。
这个算法的逻辑比较简单，效果也比较一般。

我设计的第二种规则算法被我称为“Attack"，因为这个算法考虑了胶囊的存在，对于吃豆人的行为逻辑进行了更详细的判断，同时还考虑了胶囊的存在。（具体见AgentControl.py的attack_pacman()函数。
总结而言，这个算法在距离鬼很近的时候选择全力逃跑，距离鬼很远的时候选择完全吃豆，距离鬼适中的距离时根据当前形式进行判断，在不靠近鬼的情况下可以选择吃豆子；同时，当吃豆人出于无敌状态时，吃豆人将考虑吃豆子、吃鬼，同时不再惧怕鬼。

Minimax搜索算法

然后我选择了著名的对抗搜索算法minimax搜索，来解决和鬼的对抗问题。
首先我写了minimax的朴素算法，也是从“鬼”和“食物”两方面来考虑价值。（具体见minimax_search()函

数和对应的max_search()、min_search()函数，奖励函数记录在了Parameter.py中）
我发现，自己修改不同的奖励值，吃豆人的行为智能特点也不太一样。这说明了一个模型中参数的重要性。但是这样的minimax搜索算法的权值函数需要自己来尝试并改变，因此对问题的准确判断是一个前提。

随后我写了改进版本，采用了著名的alpha-beta剪枝算法。（具体见minimax_search_with_alphabeta()函数和对应的min_search_upgraded()和max_search_upgraded()函数）效果上，两者效果是等价的，alpha-beta是一个提升速度的优化。

Q方法和Sarsa方法
Q-learning是强化学习中一种基础的算法，Sarsa算法则是它的'同策略'改进。这两个算法是基于“概率”和“试错”的学习算法，在每一次尝试后根据这次尝试的好坏更新选择的权值，这样可以让选择更优；同时算法保留一部分选择非最优解的策略（比如我采用了epsilon-贪心方法），有助于算法走出“局部最优解陷阱”。
算法的原理是：开始随机化最优策略，然后通过每一步的收益调整不同状态下不同策略的值(S,A')，使得在一次次尝试中，吃豆人会根据不同尝试的结果，逐渐找到取得分数最高的方法，即最优解。
这种基础的强化学习方法不需要我们教吃豆人如何去吃，它会自己渐渐尝试到更优的方法。
基础的Q-learning或是Sarsa都需要通过Q表格来存下学习的策略，而Q表格的大小Qsize=S*A，其中S是状态数，A是策略数。
在完整的地图中，记录完整的状态数是难以接受的，记录所有豆子是否被吃掉的完整状态，其数量更是达到了2^num（num为豆子的数量），这在大地图上将会是一个超越了存储能力的。
因此，单独的Q方法和Sarsa算法只能解决子问题Sub1.和子问题Sub.3。
对于这种情况，DQN（深度Q学习网络）和MCTS（蒙特卡洛树）算法结合可以不需要存储所有的情况，这样将可以更优秀的解决本问题。但考虑到我的时间（目前的水平）不够，所以我没有实现。

I Q方法
考虑到Python的计算速度，对于Q方法而言，我选择了联合C++和python来训练/使用展示。具体而言，我在更简化的C++代码中设计了同样的地图和鬼的逻辑，这样我就可以达到同样的效果。
C++的代码在是这个方法的一部分（./RL_C++/Q_Learning.cpp），训练结果则存放于/qtable文件夹中。同时，为了更好的接受Qtable，我设计了Qtable.py (其中有Qtable类和对应函数)来更好的在python中使用结果。

Q方法是异策略的时序差分学习方法。Q学习在更新Q表格的时候，它用到的是Q值Q(S',A)对应的动作。它不一定是下一个会执行的实际动作，因为我们下一个实际执行的动作可能是探索。Q学习默认的下一个动作不是通过行为策略来选取的，Q学习直接看Q表格，取它的最大化的值，它默认A'为最佳策略选取的动作。

在C++框架代码的基础测试中，在小地图中会有50%的性能提升（面对1个鬼），在大地图中有1000%的性能提升（面对1个鬼），或是有150%的性能提升（面对2个鬼）。

II Sarsa算法

Sarsa 是一种同策略（on-policy）算法，它优化的是它实际执行的策略，它直接用下一步会执行的动作去优化Q表格。同策略在学习的过程中，只存在一种策略，它用一种策略去做动作的选取，也用一种策略去做优化。

Sarsa算法(./RL-C++/Sarsa.cpp)的效果优于Q方法，这符合两种算法的特点。Sarsa相比Q-Learning更为谨慎

III 接口设计

如何快速的读取Q-table，是一个独立的问题。

C++生成的Qtable是一个有许多行的文件，每一行有若干个数，最后一个数是值，前面若干个数（在本游戏的Q方法中有四个数）作为索引。

我尝试了基于单哈希表的单重索引，以及多索引的多重索引，结果多索引的速度会比单索引更快。同时，我才用pickle库优化了输入，使得程序第二次输入时可以借助pickle更快的读取数据。

算法效率统计

因为Q-Learning和Sarsa的状态数限制，因此它们仅能在小地图中完成任务，在大地图中，我只为它们设计躲避鬼的任务。因此表格如此设计，Q-Learning/Sarsa和Rule-based Search /Minimax Searchde任务是不同的。

	Rule-based search	Minimax(Alpha-Beta)	Q-Learning(ϵ-greedy) 500000 Train	Sarsa(ϵ-greedy) 500000 Train
标准地图+单只鬼（通关率/时间）	100%Pass/512.82s (250s-1000s)	100%Pass/159.8s (140s-190s)	/	/
标准地图+单只鬼（存活回合）	/	/	860.37 Turn(Average) (Random: 44.2 Turn)	1025.41 Turn(Average) (Random: 44.2 Turn)
标准地图+两只鬼（通关率/平均得分）	52%Pass/1067.41s (450s-1500s)	97%Pass/184.33s (160s-240s)	/	/
标准地图+两只鬼（存活回合）	/	/	140.97 Turn (Random: 21.5 Turn)	156.74 Turn (Random: 21.5 Turn)
迷宫地图+单只鬼（通关率/时间）	100%Pass/512.82s (250s-1000s)	100%Pass/159.8s (140s-190s)	/	/
迷宫地图+单只鬼（存活回合）	/	/	1038.45 Turn (Random: 48.1Turn)	1392.50 Turn (Random: 48.1Turn)
迷宫地图+两只鬼（通关率/时间）	60%Pass/995.87s (420s-1350s)	100%Pass/154.75s (135s-180s)	/	/
迷宫地图+两只鬼（存活回合）	/	/	167.14 Turn (Random: 23.4 Turn)	179.31 Turn (Random: 23.4 Turn)
小地图+单只鬼（通关率/时间）	45%Pass/196.27s (120s-250s)	100%Pass/45s	75%Pass/73.48s	82%Pass/89.95s

这个表格符合我的预期。从结果来看，在大地图中，Minimax>Rule-based，Sarsa>Q-Learning；在小地图中，Minimax>>Sarsa>>Q-Learning>>Rule-based。
这和算法的特性有关。Sarsa更为保守，会更多的探索安全的策略，因此它的平均存活回合较长；Q-Learning则更为激进，但它也可以探索到更多的状态。
但Sarsa和Q-Learning均保留了随机的成分，这让它们在学习时可以学到新的状态，但也会让它无法达到“始终存活”的目的（否则将不再学习）。
Minimax效果远好于规则算法，这是正常的，因为Minimax搜索更为灵活，而且会根据对手来进行针对。
小地图的结果令我意外。Minimax的效果远好于QL和Sarsa，说明比起训练，告诉它奖励函数效果远远更好。但对一些更难获得奖励函数的问题而言，教学更少的优势是难以替代的。

可能将来会尝试的算法（未来展望）
第三阶段已经是本次课程设计的最终阶段，但仍有些算法我希望以后可以设计并运用在这个游戏之中。出于时间限制，我实现的算法仍不够多，也不够优秀。~~我好菜啊~~

正如之前所提到的，以下为可能的改进方向：
7-I DQN (Deep Q-Learning Network)
运用神经网络来调试出Q-Learning中的奖励函数，而不是人为设计，来提高性能。

7-II 蒙特卡洛树搜索(MCTS)
和“Q方法”类似，通过“抽样”来找到不同走法（行动）的奖励，从而进行行动。

第三阶段总结：
第三阶段是一个实际的对抗问题，因此这一阶段是最有趣的部分。
首先我扩展了游戏，实现了更复杂的交互，这对提升我的程序设计能力是有益的。
随后，我尝试了许多种不同的算法。我尝试了（在制作第二阶段的同时）学习的基础强化学习算法，尝试了不同的规则算法、对抗搜索算法（以及剪枝优化）。
不同的算法特点不同，能解决的问题也不同。规则算法需要人为总结经验，再总结给程序；搜索算法基于人的判断设定奖励函数，让程序得以进行“最优化任务”；蒙特卡洛/时序差分算法基于尝试来逐步优化自己的行为。
我仍然像第二阶段一样，将问题拆分成了两个子问题，虽然这次我不是按照子问题一个个解决的，但有些算法只能解决子问题让我意识到某些算法的局限性。

写于最后：
总的来说，做完了大作业的现在的我是惬意的。看到成品本身、看到设计的AI算法的智能，回忆起逐渐完成这个大作业的日子，让我有了一种非常的。都让我完整经历的一次学习——无论是对于游戏设计而言还是智能设计而言。
我觉得自己完成了第二阶段的期许，“我希望在第三阶段可以圆满的完成这个游戏、这个大作业，也可以入门人工智能算法和各类算法。”。
这段旅程到了尾声了。最后，像设计了这门课、设计了这些大作业的吴震老师、黄书剑老师、四位助教表示衷心的感谢！ヾ(◍°∇°◍)ﾉﾞ