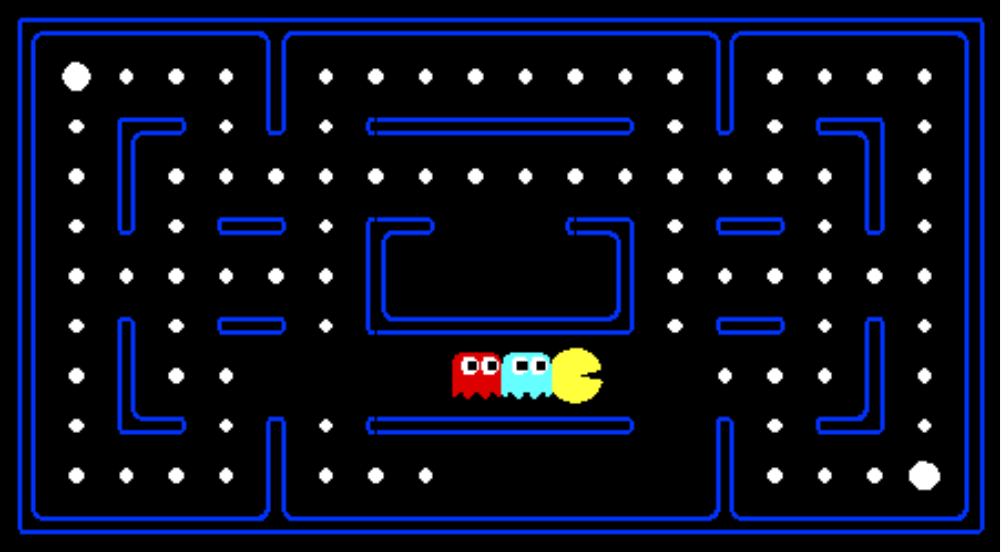
**实验二：吃豆人（对抗搜索）**



**吃豆人与幽灵**

**1. 简介**

在本次实验中，你将为经典版包含幽灵的吃豆人游戏设计一个智能体。在此过程中，你将尝试实现Minimax和Alpha-Beta搜索以及评估函数设计。

本次实验包含如下代码文件：

|  |  |  |
| --- | --- | --- |
| **需要编写的文件** | | |
| multiAgents.py | 所有的搜索智能体都将在此文件中。 |  |
| 需要查看的文件 | | |
| pacman.py | 运行吃豆人游戏的主文件。该文件包含了GameState类，它将在游戏中被广泛运用。 |  |
| game.py | 该文件实现了吃豆人游戏的运行逻辑，包含了像AgentState，Agent，Direction，Grid等几个起到支持作用的类。 |  |
| util.py | 该文件包含了实现搜索算法需要的数据结构。 |  |
| 可忽略的支撑文件 | | |
| ghostAgents.py | 该文件控制幽灵的智能体。 |  |
| graphicsDisplay.py | 该文件实现吃豆人游戏的图形界面。 |  |
| graphicsUtils.py | 该文件为吃豆人游戏的图形界面提供支持。 |  |
| keyboardAgents.py | 该文件实现通过键盘控制吃豆人。 |  |
| layout.py | 该文件包含了读取游戏布局文件和保存布局内容的代码。 |  |
| textDisplay.py | 该文件为吃豆人游戏提供ASCII码形式展现的图形。 |  |

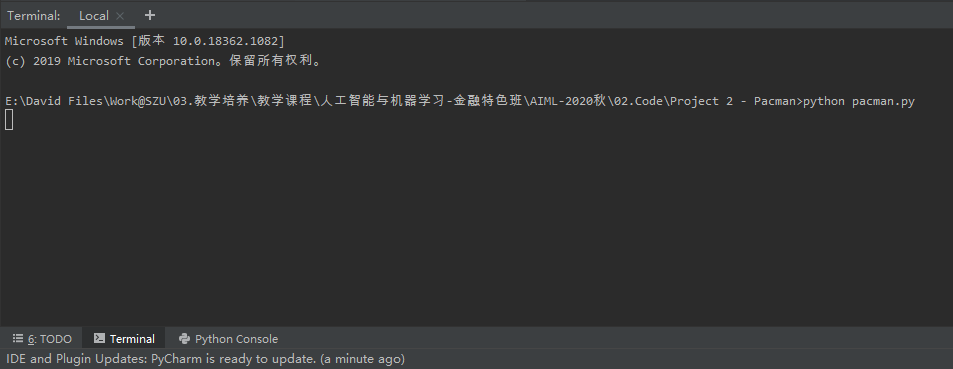
**需要编辑和提交的文件：**在本次实验中，你需要在文件multiAgents.py中补充代码。提交该文件时文件中应包含你编写的代码和注释。

**学术不端问题：**我们会对你的代码和班上其他同学的代码进行对比检查。如果你抄袭了别人的代码，或者修改别人代码的一小部分后提交，我们是会知道的。代码查重程序几乎不会放过这种不诚信的行为，所以请不要试探查重程序的底线。我们相信你们提交的实验都是自己完成的，请不要让我们失望。如果你出现了学术不端行为，我们会在自己的能力范围内进行严惩。

**题0：吃豆人初体验**

首先体验一下经典的吃豆人游戏（通过键盘方向键控制），调用代码为：  
 *python pacman.py*

具体操作：用Pycharm打开软件包，点击界面左下底部的Terminal，路径中输入命令。



现在，运行在multiAgent.py文件中已经提供的ReflexAgent(简单反射智能体，不关注历史信息)：  
 *python pacman.py -p ReflexAgent*  
 留意到它在简单的地图排布上效果也不太好：

*python pacman.py -p ReflexAgent -l testClassic*

详细查阅multiAgents.py文件中关于ReflexAgent的类函数，确保已理解该类的代码流程细节。

**题1：Reflex智能体**

改进multiAgents.py文件中的ReflexAgent类以更好地玩吃豆人游戏。代码中提供的**ReflexAgent类代码**包含了一些有帮助的方法的例子，而这些方法是可以通过**查询GameState类**获取相关信息。**一个强的Reflex智能体能够同时考虑食物和幽灵的位置以取得好的表现。你改进的Reflex智能体应该能轻松可靠地清空testClassic布局**。使用以下命令进行验证：  
 *python pacman.py -p ReflexAgent -l testClassic*

**让你自己的Reflex智能体在默认的mediumClassic布局运行，运行时可以设置1个或2个幽灵，同时关闭动画以提升加速显示。** *python pacman.py –frameTime 0 -p ReflexAgent* ***-k 1***  
 *python pacman.py –frameTime 0 -p ReflexAgent -k 2*

你的智能体是如何取得成功的？**智能体在默认布局里会经常因为碰到幽灵而死去，除非你的评估函数很好。**  
 **注意：**作为特征，**尝试使用重要评估数值（例如到食物的距离）的倒数，而不是这些数值本身**。另外你要编写的评估函数**评估的是状态-行为的配对**；实验的后续部分中，**你将要评估状态。** **可选设置：**默认情况下幽灵位置随机；为了获得乐趣，你可以使用命令 -g DirectionalGhost 使游戏中的幽灵更聪明和有方向性。如果程序的随机性让你无法判断你的智能体是否有改善，你可以使用命令 -f 使随机化情况被固定为某一种。你也可以通过命令 -n 来让游戏运行多次。使用命令 -q 关闭图形化界面使游戏更快运行。

*评分：***我们会将你的智能体在openClassic布局上运行10次**。如果你的智能体运行超时，或者从未取胜就会得到0分。如果你的智能体最少5次胜利就能得到1分，胜利10次可得到2分。如果游戏平均得分超过500分就能额外得到1分，**平均得分超过1000分就能额外得2分。**

*python pacman.py -p ReflexAgent -l openClassic*

**题2：Minimax**

现在你要在multiAgents.py文件里自带的**MinimaxAgent类中编写一个对抗性的搜索智能体**。你编写的Minimax智能体需要能应对至少一个幽灵，因此你需要编写一个比课上讲到的算法更加通用的一个算法（多个幽灵情况，实验不做强制要求，有兴趣的同学可以尝试）, **你的Minimax树要有多个最小层（一个最小层对应一个幽灵）对应每一个最大层。**

你的代码应该也要**能将游戏中的Minimax树扩展到任意深度**。通过提供的**self.evaluationFunction（其默认指向scoreEvaluationFunction）**为Minimax树的叶子结点记录分数。**MinimaxAgent类是MultiAgentSearchAgent类的子类，MultiAgentSearchAgent类为MinimaxAgent类提供了self.depth和self.evaluationFunction的访问**。**你的代码要确保在合适的地方指向这两个变量，因为这两个变量要对命令行选项做出密集的反应。**

**重要：**单个搜索层被定义为**包含吃豆人单次移动和所有幽灵的反应**。因此**深度为2的搜索将包含吃豆人和各个幽灵各两次的移动。**

**提示：**

1. Minimax算法的正确实现在一些测试中**会导致吃豆人输掉游戏**。这不是问题：只要行为正确，测试就能通过。

2. 这一题吃豆人游戏测试所使用的**评估函数已经写好了（self.evaluationFunction）。**你不应该修改函数，但要意识到这一题评价的是**状态**而不是评价Reflex智能体时的行动。**Look-ahead智能体评价的是未来的状态**，然而Reflex智能体评价的是当前状态下的行动。

3. 在minimaxClassic布局游戏的初始状态中，**Minimax树中深度为1、2、3、4的结点的数值分别是9、8、7、-492**.要留意的是尽管深度为4的Minimax树的预测结果极差，但你的**Minimax智能体经常会赢（出题人测试得66.5%胜率）。**

*python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4*

4. 吃豆人一直都是智能体0，所有智能体按照智能体编号升序先后移动。

5. 该题中所有游戏状态都应该为GameState类实例，或者传递到getAction函数或者通过GameState.generateSuccessor生成。在实验中，你将不会对简化的状态进行概括。

6. 在像openClassic和默认的mediumClassic的更大的游戏空间里，你会发现吃豆人很难死亡，但也很难取胜。智能体常常会在一个区域翻来覆去，没有进展。智能体甚至有可能在一个点周围翻来覆去，但不吃那个点，因为它不知道吃掉那个点之后要怎么走。不用担心，如果你看到了这样的行为，题5会解决掉这些问题。

7. 当吃豆人发现它将不可避免地死亡时，它会尝试尽快结束游戏因为不断的存活罚分。有时候，这个是跟随机幽灵有关的错误做法，但Minimax智能体一直都假定最坏情况：

*python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3*

你要确保自己明白为什么在这样的情况下吃豆人会冲向最近的幽灵。

**题3：Alpha-Beta剪枝**

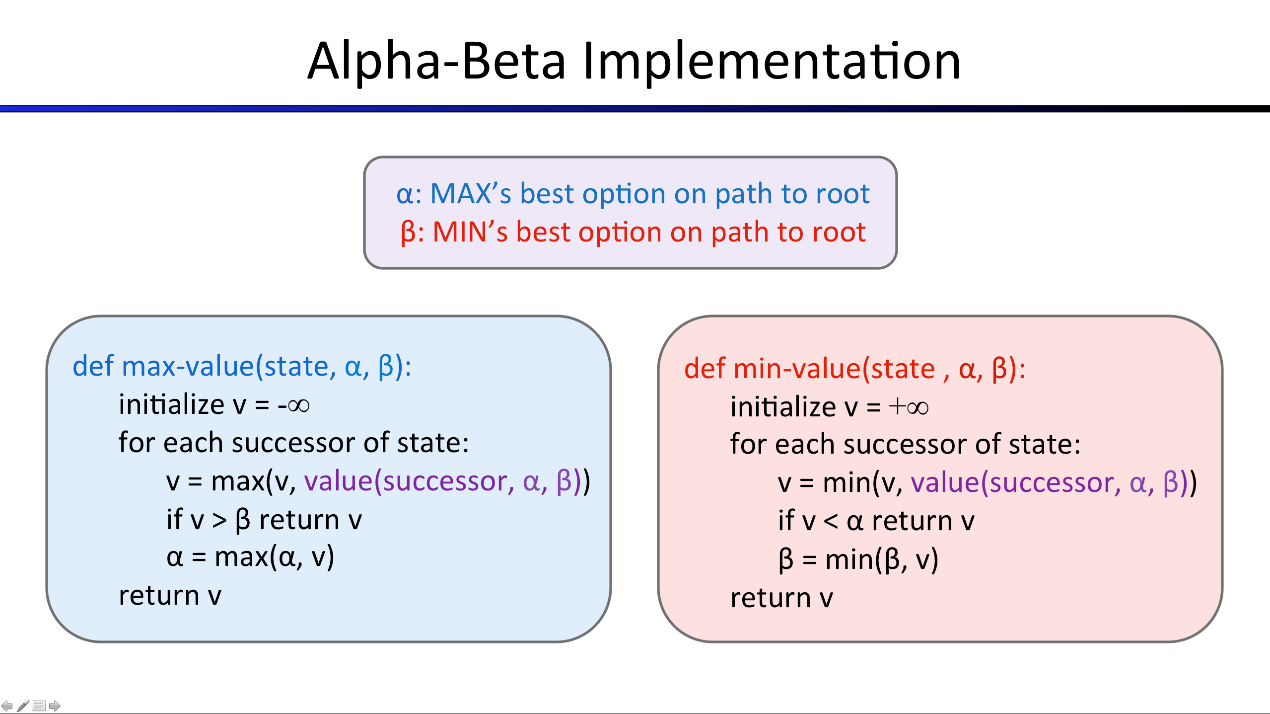
在AlphaBetaAgent类中编写一个**使用Alpha-Beta剪枝的新智能体**，使得Minimax树的访问更高效。同样的，你的算法**要比课上的伪代码更加通**用，所以挑战的一部分是要将Alpha-Beta剪枝逻辑**适当推广到多个最小化智能体**（多个幽灵情况，实验不做强制要求，有兴趣的同学可以尝试）。

你应该**会看到速度的提升**（也许深度为3的Alpha-Beta树的访问会跟深度为2的Minimax树一样快）。在**smallClassic布局中深度为3的树的理想情况下，每次移动的运行耗时不慢于几秒钟。**

*python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic*

在AlphaBetaAgent类中生成的Minimax生成树的值应该与在MinimaxAgent类中生成的Minimax生成树的值相等，即使根据不同的“同分决胜”方式选择的行为可能会不同。同样的，在minimaxClassic布局游戏的初始状态中，M**inimax树中深度为1、2、3、4的结点的数值分别是9、8、7、-492.**

下面是你要本题要实现的算法对应的伪代码。



**题4：评价函数（可选，不做强制要求）**

**在自带函数betterEvaluationFunction中为吃豆人编写一个更好的评价函数**。评价函数应该**要评价状态**，而不是像Reflex智能体所做的那样去评价行动。你可以为了评价使用任何工具进行处理，**包括使用经典搜索算法代码**。在深度为2的搜索中，**你的评价函数应该能在只有一个随机生成的幽灵、布局为smallClassic的游戏中超过一半机率清空布局上所有的豆子，并且仍然能取得一个合理的评分**（**吃豆人取胜时平均得分在1000分左右时题目才能得满分**）。

**评分：**将会让你的智能**体在smallClassic布局上运行10次**。我们会通过以下方式对你编写的评价函数进行打分：

1. 在没有超时的情况下至少赢一盘，得1分。任何智能体不满足这一点标准的得0分。

2. 赢最少5次加1分，10次全赢加2分。

3. 平均得分达到500分者再加1分，达到1000分者则再加2分（包括落败的游戏的得分）。

4. 评分过程中每一局游戏平均用时小于60秒的加1分。

5. 只有赢最少5次才能得到平均得分和运行时间方面的附加分。

**提示：**

**编写的评价函数是使用多个特征的线性组合。也就是，计算特征在你认为重要的状态下的值，然后通过权重相加得到最终结果，权重值可以根据特征的重要性来设定。**