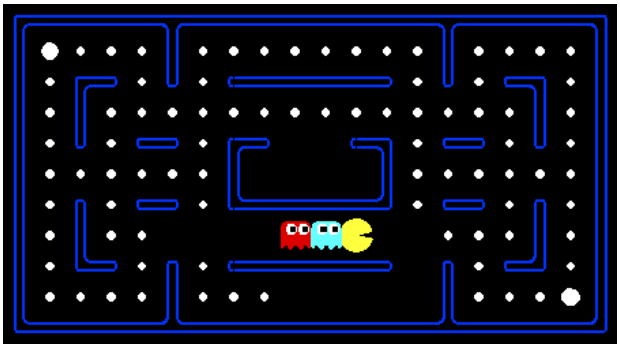
10 Multi-Agent Search II



Pacman, now with ghosts.

本节实验目标：练习多智能体搜索的剪枝算法和expectimax

报告说明（重要）：本部分“多智能体搜索”包含两周内容，本节课为第二周内容，请留意该文档目录中标有“报告应实现任务[分值]”的部分，请在实现后及时保存文件和运行结果截图，在本次课程报告中提交。

Introduction
欢迎来到 Multi-Agent Pacman 世界
上节课参考实现
    Question 1: Reflex (反射) Agent 报告应实现任务[1分]
    Question 2: Minimax 报告应实现任务[1分]
Question 3: Alpha-Beta Pruning 报告应实现任务[1分]
    任务说明（建议做任务时看一下该文档的提示部分）
    提示
    测试
Question 4: Expectimax（建议做任务时看一下该文档的提示部分）报告应实现任务[1分]
    测试
Question 5: Evaluation Function 报告应实现任务[1分]
报告4 “ Multi-Agent Search”提交说明

Introduction

在本堂实验课，你将体验经典版本的吃豆人游戏，它加入了幽灵ghosts。

像之前一样，这个项目包括一个autograder为你的答案评分。这可以使用以下命令对所有问题运行：

```
python autograder.py
```

它可以通过以下方式针对一个特定问题（例如 q2）运行：

```
python autograder.py -q q2
```

或者针对一项特定的测试运行：

```
python autograder.py -t test_cases/q2/0-small-tree
```

默认情况下，autograder使用 `-t` 选项显示图形，但不使用 `-q` 选项。你可以使用 `--graphics` 标志强制使用图形，或使用 `--no-graphics` 标志强制不使用图形。

此项目的代码可在这里下载，包含以下文件：

你将编辑的文件：	
multiAgents.py	multiAgents的所在地
你可能想查看的文件：	

你将编辑的文件：	
pacman.py	运行 Pacman 游戏的主文件，描述了一个你要用到的 Pacman **GameState** 类型。
game.py	Pacman 世界如何运作背后的逻辑，描述了如 **AgentState**、Agent、Direction 和 Grid 等类型。
util.py	一些用于实现搜索算法的数据结构。你不需要在此项目中使用这些，但可能会发现此处定义的其他函数很有用。
你可以忽略的支持文件：	
graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	用键盘控制 Pacman 的程序
layout.py	Code for reading layout files and storing their contents
autograder.py	Autograder
testParser.py	Parses autograder test and solution files
testClasses.py	General autograding test classes
test_cases/	Directory containing the test cases for each question
multiagentTestClasses.py	Specific autograding test classes

你将在 `multiAgents.py` 中实现部分内容，autograder 用来评判你的技术正确性，请不要更改代码中提供的任何函数或类的名称，否则 autograder 无法正确运行。

欢迎来到 Multi-Agent Pacman 世界

首先，试着手动控制一下Pacman（上下左右箭头或WASD，**按Q能让吃豆人在移动中停下来**）：

```
python pacman.py
```

然后，可以体验 `multiAgents.py` 里实现的 `ReflexAgent`，它不用你手动控制：

```
python pacman.py -p ReflexAgent
```

注意，它的表现很一般，即使在下面这个简单的地图也是如此：

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

上节课参考实现

Question 1: Reflex (反射) Agent 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q1 --no-graphics 的运行结果，获得评分器给的4分中至少3分，这道题才算得它占的1分。**

改进 `multiAgents.py` 中的 `ReflexAgent`，代码里的 `ReflexAgent` 提供了一些有用的方法示例，这些示例可以查询 `GameState` 以获取信息。一个还不错的反射Agent必须同时考虑食物位置和幽灵位置才能表现良好。**改进后**，你的agent应轻松可靠地吃掉上面提到的 `testClassic` 环境中的豆子：

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

在默认的 `mediumClassic` 迷宫上使用一个或两个幽灵（-k 1 或 -k 2）并关闭动画以加快显示速度（—frameTime 0；注：如是—frameTime 1 则会特别慢）尝试你改进的反射Agent：

```
python pacman.py --frameTime 0 -p ReflexAgent -k 1
```

```
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

你的Agent表现如何？除非你写的评估函数非常好，否则它可能会经常在有两个幽灵的`mediumClassic`迷宫中game over。

参考实现：

```
def evaluationFunction(self, currentGameState, action):
    successorGameState = currentGameState.generatePacmanSuccessor(action) # 在当前状态后采取一个行动后到达的状态
    newPos = successorGameState.getPacmanPosition() # 下一个状态的位置 （x，y）
    newFood = successorGameState.getFood() # 下一个状态时环境中的食物情况 (TTTFFFFFT......)
    newGhostStates = successorGameState.getGhostStates() # 下一个状态时幽灵的状态
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates] # 吃了大的白色食物（能量点）后，白幽灵的剩余持续时间

    "*** YOUR CODE HERE ***"
    newFood = newFood.asList() # [(x1, y1), (x2, y2), ...] 剩余食物的位置
    ghostPos = [(G.getPosition()[0], G.getPosition()[1]) for G in newGhostStates] # [(x1, y1), (x2, y2)] 幽灵一（和幽灵二）的位置
    scared = min(newScaredTimes) > 0 # 是否幽灵是出于白色状态（可以被agent吃掉的状态）

    if not scared and (newPos in ghostPos): #如果agent在下个位置碰到的是正常的幽灵，那游戏结束了，gg
        return -1.0

    if newPos in currentGameState.getFood().asList(): # 否则，如果下一个位置能吃到食物，这个状态-动作对得一分
        return 1

    # 注： key=lambda fPos 即对newFood列表里的每个值，计算与下个位置的曼哈顿距离，以这个作为到食物距离的排序依据
    closestFood = sorted(newFood, key=lambda fPos: util.manhattanDistance(fPos, newPos)) # 下个位置离所有食物预估距离的从近到远排序
    closestGhost = sorted(ghostPos, key=lambda gPos: util.manhattanDistance(gPos, newPos)) # 下个位置离所有幽灵预估距离的从近到远排序

    # 状态-动作对 得分：离食物的最近预估距离的倒数 减去 离幽灵的最近预估距离的倒数
    # 动机：越好的状态得分越高，离最近的食物越近越好（取距离倒数），离最近的幽灵越远越好（取距离倒数，前面再加个减号）
    return 1.0 / util.manhattanDistance(closestFood[0], newPos) - 1.0 / util.manhattanDistance(closestGhost[0], newPos)
```

Question 2: Minimax 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q2 --no-graphics 的运行结果，获得评分器给的5分中，这道题才算得它占的1分。**

为了能在任意数量的幽灵中夹缝求生，你必须编写一个比你之前在课堂上看到的一对一对抗的情况更通用的算法。特别是，你的 minimax 树中，每个 max 层下将有多个 min 结点（每个幽灵是一个Min）。

实际中的幽灵可能会部分随机地行动，但 Pacman 的极小极大算法假设最坏的情况（即幽灵的行动决策会最小化我们agent的收益）。**实现后**，可以运行以下代码看看效果：

```
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3 --frameTime 1
```

参考实现：

```
def getAction(self, gameState):
"""
    Returns the minimax action from the current gameState using self.depth
    and self.evaluationFunction.根据当前的游戏状态，返回一个根据minimax值选的最佳行动

    Here are some method calls that might be useful when implementing minimax.
以下的一些函数调用可能会对你有帮助
    gameState.getLegalActions(agentIndex):
    Returns a list of legal actions for an agent返回一个agent（包括吃豆人和幽灵）合法行动（如不能往墙的地方移动）的列表
    agentIndex=0 means Pacman, ghosts are >= 1

    gameState.generateSuccessor(agentIndex, action):
    Returns the successor game state after an agent takes an action一个agent采取行动后，生成的新的游戏状态

    gameState.getNumAgents():
    Returns the total number of agents in the game获取当前游戏中所有agent的数量

    gameState.isWin():
    Returns whether or not the game state is a winning state判断一个游戏状态是不是目标的胜利状态

    gameState.isLose():
    Returns whether or not the game state is a losing state判断一个游戏状态是不是游戏失败结束的状态
    """
"*** YOUR CODE HERE ***"
    GhostIndex = [i for i in range(1, gameState.getNumAgents())] # 获取幽灵的索引

    def terminate(state, depth): # 定义一个是否搜索终止的判断函数
        return state.isWin() or state.isLose() or depth == self.depth #self.depth已经到达搜索深度

    def min_value(state, depth, ghost):  # minimizer，这里ghost指幽灵的索引，也作为参数输入，因为更通用的minimax可以handle多个幽灵的情况

        if terminate(state, depth): # 如果到了搜索终止的条件了，返回幽灵结点的 min_value（评估函数值）
            return self.evaluationFunction(state)

        v = 10000000000000000 # 近似正无穷
        for action in state.getLegalActions(ghost):
            if ghost == GhostIndex[-1]: # 如果是最后一个幽灵了，这层深度的搜索就完成了，返回ghost采取行动后下一层的吃豆人结点的最大状态值
                # state.generateSuccessor(ghost, action) 对应课本伪代码 successor of state:
                v = min(v, max_value(state.generateSuccessor(ghost, action), depth + 1))
            else: # 当前最佳行动的幽灵(所有幽灵中min最小的那个)（这里有个递归，好好品味一下...）这个v存储的是计算到当前索引幽灵中行动最好的状态值（即对吃豆人最不好的一个幽灵行动）
                v = min(v, min_value(state.generateSuccessor(ghost, action), depth, ghost + 1))
        return v

    def max_value(state, depth):  # maximizer

        if terminate(state, depth): # 如果到了搜索终止的条件了，返回吃豆人结点的 max_value（评估
```

```
函数值)
            return self.evaluationFunction(state)

        v = -10000000000000000
        for action in state.getLegalActions(0): # 吃豆人的合法行动
            v = max(v, min_value(state.generateSuccessor(0, action), depth, 1))
        return v

    # gameState.generateSuccessor(0, action), 0, 1) =>  (下一个游戏状态即min结点，从深度为0,
第一个幽灵开始算它的行动)
    # 对当前状态吃豆人(max)的每个合法行动，计算minimax value (next agent is min)
    result = [(action, min_value(gameState.generateSuccessor(0, action), 0, 1)) for acti
on in
           gameState.getLegalActions(0)]

    result.sort(key=lambda k: k[1]) # result = [(action1, value1), (action2, value2),
...],  lambda k: k[1] 指按 value从小到大

    return result[-1][0] # 取状态值最大的(action, value)里的action
```

Question 3: Alpha-Beta Pruning 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q3 --no-graphics 的运行结果，获得评分器给的所有5分，这道题才算得它占的1分。**

任务说明（建议做任务时看一下该文档的提示部分）

在 `AlphaBetaAgent` 中创建一个使用 alpha-beta 剪枝来更有效地探索minimax搜索树的新agent。同样，你的算法将比课堂中的伪代码更通用，本题的部分挑战是将 alpha-beta 剪枝逻辑适当地扩展到多个minimizer agents（多个幽灵）。

实现后，你应该会看到加速的搜索过程（也许搜索深度为3的 alpha-beta剪枝算法将与深度 2的minimax算法一样快）。理想情况下，`smallClassic`上的深度为3的搜索应该每次移动只需几秒钟或更快。

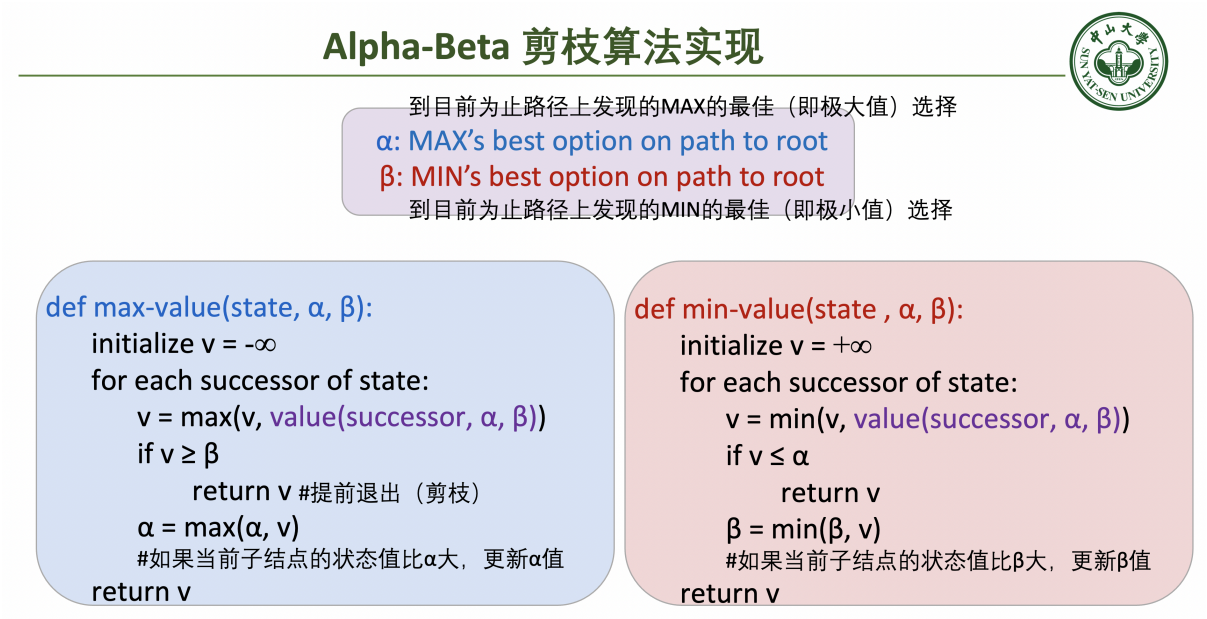
```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

`AlphaBetaAgent` 的极小值应该与 `MinimaxAgent` 的极小值相同，尽管它选择的动作可能会因不同的平局行动而有所不同。同样，对于深度 1、2、3 和 4，`minimaxClassic` 迷宫中初始状态的极小极大值分别为 9、8、7 和 -492。

提示

Autograder 会检查你的代码以确定它是否探索了正确数量的状态，所以执行 alpha-beta 剪枝而不重新排序子结点非常重要。换言之，应始终按照 `GameState.getLegalActions` 返回的顺序处理子状态（这里不用关注剪枝的顺序）。同样，不要过度调用 `GameState.getNextState`。

下面是课堂上相关的伪代码：



测试

测试和调试你的代码（动画展示吃豆人的行为，可能可以帮助debug），运行：

```
python autograder.py -q q3
```

要在没有图形动画的情况下运行它（会更快），运行：

```
python autograder.py -q q3 --no-graphics
```

一个正确执行 alpha-beta 剪枝的算法会导致 Pacman 没有完成某些吃豆任务。这不是问题，因为它是正确的行为，它会通过autograder的测试。

Question 4: Expectimax（建议做任务时看一下该文档的提示部分）报告应实现任务 [1分]

- **说明：该小题截图只需截 python autograder.py -q q4 --no-graphics 的运行结果，获得评分器给的所有5分，这道题才算得它占的1分。**

Minimax 和 alpha-beta 很棒，但它们都假设你正在与做出最佳决策的对手对抗。但情况并非总是如此。在这个问题中，你将实现 `ExpectimaxAgent`，这对于建模可能做出次优选择的agents的概率行为很有用。

与前面两问的算法一样，这些算法的美妙之处在于它们的普遍适用性。为了加快你自己的开发，我们提供了一些基于通用树的测试用例。在实现 `ExpectimaxAgent` 过程，你可以使用以下命令在小型游戏树上调试：

```
python autograder.py -q q4
```

建议对这些小型且易于管理的测试用例进行调试，这将帮助你快速找到错误。

一旦你的算法在小型搜索树上运行，你就可以在 Pacman 中观察到它的成功。随机幽灵当然不是最优的极小极大agents，因此使用极小极大搜索对其进行建模可能不合适。 `ExpectimaxAgent` 将根据你的agent模型对幽灵的行为方式进行预期，而不是采取所有幽灵行动状态值最小的一个。为了简化你的代码，假设幽灵只在其 `getLegalActions` 中随机选择一个行动。

要查看 `ExpectimaxAgent` 在 Pacman 游戏中的行为，请运行：

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

你现在应该观察到吃豆人在与幽灵近距离接触时表现出一种更加随意的行动方法。特别是，如果吃豆人意识到它可能被困住，但可能也能逃跑去抓几块食物，它至少会尝试一下。试试这两种情况的结果：

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

你应该会发现你的`ExpectimaxAgent`赢了大约一半的测试(如果想看慢动作效果，上面的命令可以去掉后面的 -q -n 10，而加上—frameTime 1 (注意，—是两个减号-))，而你的`AlphaBetaAgent`总是输。确保你理解为什么这里的行为与 minimax 情况不同。

一个正确执行 expectimax的算法仍然会导致 Pacman 没有完成某些吃豆任务。没关系，只要它是正确的行为，它就会通过autograder的测试。

测试

测试和调试你的代码（动画展示吃豆人的行为，可能可以帮助debug），运行：

```
python autograder.py -q q4
```

要在没有图形动画的情况下运行它（会更快），运行：

```
python autograder.py -q q4 --no-graphics
```

Question 5: Evaluation Function 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q5 --no-graphics 的运行结果，获得评分器给的所有6分的至少4分，这道题才算得它占的1分。**

在提供的函数`betterEvaluationFunction`中为 pacman 编写更好的评估函数，它应该评估状态，而不是像ReflexAgent评估函数那样评估动作。在深度为 2 的搜索时，你的评估函数应该有一半以上的游戏次数里搞定了带有一个随机幽灵的`smallClassic`地图，并且仍然以合理的速度运行（在比较好的情况下，Pacman 获胜时的平均得分应该在 1000 左右）。

Autograder 将在 smallClassic 地图上运行你的agent 10 次，它通过以下方式为你写的`betterEvaluationFunction`评分：

- 如果你在autograder没超时运行的情况下至少赢了一次，你将获得 1 分。任何不满足这些标准的agent都将获得 0 分。
- 至少赢得 5 次 +1，赢得全部 10 次 +2
- 平均分至少 500 分 +1，平均分至少 1000 分 +2（包括输掉比赛时的分数）
- 如果你的游戏在使用`-no-graphics`运行时在autograder程序上平均耗时不到 30 秒，则 +1。
- 平均分数和计算时间的额外积分只有在你至少获胜 5 次时才会获得。

你可以在这些条件下试用你的agent（动画展示吃豆人的行为，可能可以帮助debug）

```
python autograder.py -q q5
```

要在没有图形动画的情况下运行它（会更快），请运行：

```
python autograder.py -q q5 --no-graphics
```

报告4 “ Multi-Agent Search”提交说明

- 按照第9周和本周第10周的Question 1-5 的要求实现代码和获得运行结果截图
- **提交压缩包命名为”姓名_学号_报告序号.zip”** (如“彭振辉_2106666_报告4.zip”)
- 压缩包应**包含内容：**
  - **已实现的完整项目文件夹”Project_4_Multi-Agent_Search_full“**
    - 其中 multiAgents.py 中有Question 1-5要求的函数实现
  - **一个doc或pdf说明文档，上面需要有：**
    - 开头一段说明 “**整体实现参考 + 2-3句简要体会（如教训、思路、拓展应用等）**”，如：-

- “自行实现。挑战最大的是xxx内容，初始时报了什么错，通过什么方式解决，该部分的实现思路为xxx”
- “xxx内容参考xxx同学/xxx网址。思考不出算法思路，探究后学习到了什么方法。”

- **要求实现的5个任务的成功运行截图，说明截图对应任务。**