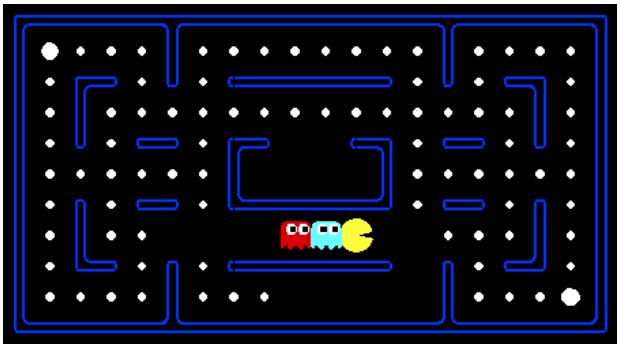
09 Multi-Agent Search I



Pacman, now with ghosts.

本节实验目标：练习多智能体搜索的极小极大算法

报告说明（重要）：本部分“多智能体搜索”包含两周内容，本节课为第一周内容，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，在下一周课程报告中一并提交。

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
```

Note: 如果你自己电脑上的 `python` 是 2.7版本（实验室的pycharm里应是3.6版本的，不用管），你可能需要调用 `python3 autograder.py` 或创建一个 conda 环境。

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：	
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。

你将编辑的文件：	
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
```

在 `multiAgents.py` 中查看ReflexAgent的代码，确保你理解它在做什么。

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。**

任务说明（建议做任务时看一下该文档的提示部分）

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。

提示

Note: 注意， `newFood` 类中的 `asList()` 函数可能挺有用

Note: 作为特征，尝试一些值的倒数（例如与食物的距离），而不仅仅是值本身。

Note: 你要改进的评估函数会评估 “状态-动作”对 。

Note: 用print()函数查看对象的内部内容会方便你的调试，比如要查看`newGhostStates`，

Note: 你可能会发现在调试时，查看各种对象的内部内容很有用(`print(...)`)，但print一个state可能无法显示其内部内容，需要进一步调用state的方法获取内部内容。

Options: 默认的幽灵是随机行动的；你还可以使用 `-g DirectionalGhost` 来跟更聪明的幽灵斗智斗勇。如果随机性使你无法判断你的agent是否在改进，你可以使用 `-f` 固定随机种子来运行幽灵（即每场游戏幽灵都使用相同的随机选择）。你还可以使用 `-n` 连续玩多个游戏（比如10个： `-n 10` ）。使用 `-q` 关闭动画显示以快速运行大量游戏。

测试

Autograder会在 `openClassic` 迷宫中运行你的Agent 10 次。如果你的agent超时或获胜次数少于5次，你将获得 0 分。如果您的agent赢了至少 5 次，你将获得 1 分，如果你的agent赢了所有 10 场比赛，你将获得 2 分。如果你的agent的平均分数大于 500，你将获得额外的 1 分，如果大于 1000，你将获得 2 分。**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。**你可以在这些条件下尝试Question 1的agent：（动画展示吃豆人的行为，可能可以帮助debug）

```
python autograder.py -q q1
```

要在**没有图形动画的情况下**运行它（出结果会快一些），请运行：

```
python autograder.py -q q1 --no-graphics
```

不过，先不要在这个问题上花太多时间（后面再回来把玩一下），因为后面的内容更精彩。

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。**

任务说明（建议做任务时看一下该文档的提示部分）

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 `multiAgents.py` 中的 `MinimaxAgent` 中的getAction函数，请看该函数下的一些提示。**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
```

确保你理解为什么在这种（必定被吃的）情况下吃豆人会冲向最近的幽灵（实现`MinimaxAgent`后你可以观察到的现象）。

你的代码应该可以将游戏树扩展到任意深度。使用提供的`self.evaluationFunction`，默认为`scoreEvaluationFunction`，对极小极大树的结点进行评分。 `MinimaxAgent` 扩展了 `MultiAgentSearchAgent`，它可以访问 `self.depth` 和 `self.evaluationFunction`。**确保你的 minimax 代码在适当的地方引用这两个变量，因为这些变量是为响应命令行选项（如 depth=3）而填充的。**

重要提示： 单层深度的搜索被认为是一次”吃豆人移动+所有幽灵的反应“，因此深度为2的搜索将涉及吃豆人和每个幽灵移动两次。

测试

Autograder 将检查你的`MinimaxAgent`代码，以确定它是否探索了正确数量的游戏状态。这是在 minimax 的实现中检测一些非常微妙的错误的唯一可靠方法。因此，autograder会非常关注你调用`GameState.getNextState`的次数。如果你调用它的次数比必要的次数多或少比较多，autograder会识别出来。要测试和调试你的代码，可以运行（正确实现后，这个效果看起来比上面那个必定被吃的好很多）：

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

要在没有图形动画的情况下运行它（会快一些），请使用：

```
python autograder.py -q q2 --no-graphics
```

提示

- Hint: 可能需要递归地实现所需的算法。
- 正确实现了一个minimax 仍然会导致 Pacman 在某些测试中输掉。这不是问题：只要它是正确的行为，它就会通过autograder的测试。
- `self.evaluationFunction` 这部分 Pacman 测试的评估函数已经编写好了。你**不应该更改**此评估函数，但要认识到现在我们正在评估的是agent的状态而不是动作（Question 1 中ReflexAgent评估的是状态-动作对）。
- 对于搜索深度 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、-492。请注意，尽管深度 4 时 agent可能会做一些可怕的预测，你的minimax agent还是有点胜算的（想测试胜率，可以在下面代码加 -n 20 -q），实现后，试一下这个：

  ```
  python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --frameTime 1
  ```

- **吃豆人的代号总是 agent 0，并且agents（包括幽灵）按照agent索引增加的顺序依次行动。**
- minimax 中的所有状态都应该是`GameStates`，要么作为参数传递给`getAction`，要么通过`GameState.getNextState`生成。
- 在诸如 openClassic 和 mediumClassic（默认）等较大的迷宫上（改变上面代码 -l 后的选项即可尝试），你会发现 Pacman 擅长不被幽灵抓到，但不擅长获胜（吃掉所有的食物）。它经常在没有进展的情况下胡思乱想... ... 它甚至可能在不吃的情况下就在一个点旁边扭动，因为它不知道吃了那个点后它会去哪里。如果它看到此行为，请不要担心，之后我们会处理这个问题。