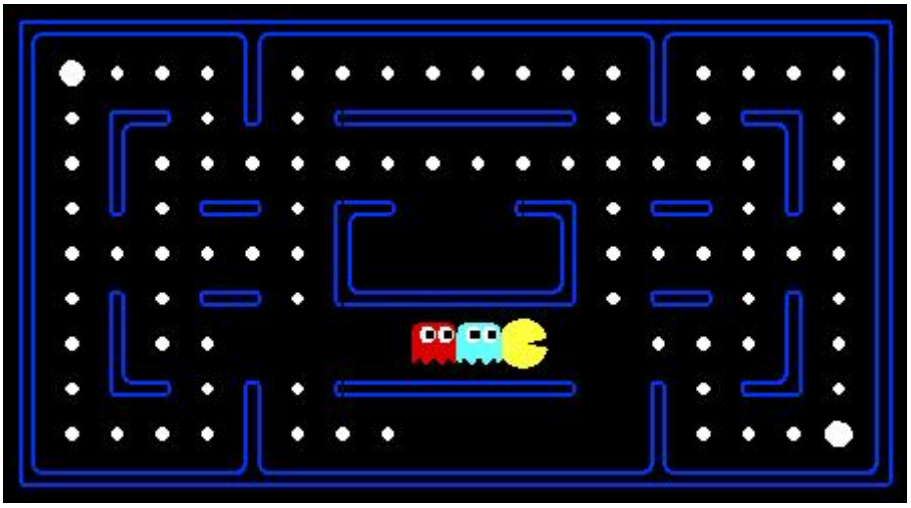
信息与计算机工程学院

人工智能导论

课程项目 2：多人搜索



1、介绍

在此项目中，你的吃豆人将与幽灵竞争。在此过程中，你将实现最小最大值和期望最大值搜索，并尝试评估函数设计。

代码库与以前的项目没有太大变化，但请从全新安装开始，而不要和项目 1 中的文件混合。

与项目 1 一样，该项目包括一个自动批改程序，供你在机器上对答案进行评分。可以使用以下命令对所有问题运行此操作：

```
python autograder.py
```

它可以针对一个特定问题运行，例如 q2，方法是：

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

它可以通过以下形式的命令为一个特定的测试运行：

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

默认情况下，使用 -t 选项，自动批改程序会显示图形，使用 -q 选项则不显示图形。可以使用 --graphics 标志强制图形，也可以使用 --no-graphics 标志强制不显示图形。

有关使用自动批改程序的详细信息，请参阅 Project 0 中的自动批改程序教程。

此项目的代码包含以下文件，这些文件以 zip 文档形式提供。

你需要编辑的文件	
multiAgents.py	在这里实现你的搜索算法
你需要查看的文件	
pacman.py	执行吃豆人游戏的主要程序，描述游戏状态
game.py	吃豆人世界运行的逻辑，有许多重要的定义，例如 AgentState，Agent，Direction 和 Grid
util.py	里面有些可以帮助实现搜索算法的数据结构

其它的文件你可以忽视。

需要编辑和提交的文件：你需要完成 multiAgents.py，将你修改的文件和自动批改程序（submission_autograder）产生的结果，以及项目报告一同提交。

此外，必要的话，你可以使用项目 1 中的函数和文件（例如 search.py 和 searchAgents.py 中的 ClosestDotSearchAgent 和 mazeDistance）来帮助你完成这个项目。假如你用到了之前项目的文件，请将它们也一道打包提交。

请不要修改或提交其它文件。

项目评估：你的代码会通过自动批改来判断其正确性，因此请不要修改代码中其它任何函数或者类，否则你会让自动批改程序无法正常运行。然而，你的解题思路和方法是你最终成绩的决定因素。必要的话，我们会查看你的代码来保证你得到应得的成绩。

学术造假：我们会查看你的代码和其它学生提交的代码是否雷同。禁止抄袭了他人代码，或只做简单修改后提交，一旦发现，成绩立马作废，而且会影响到你能否通过此课程。

寻求帮助：当你感到自己遇到了困难，请向你的同学和老师寻求帮助。小组合作、答疑时间、课堂讨论，这些都是用来帮助你的，请积极利用这些资源。设计这些项目的目的是让你更有效地理解和掌握课堂知识，学会如何将理论知识应用于实践，解决实际问题，而不是为考核而考核，或者有意刁难你，所以请尽你所能，完成它们。遇到困难时，向学生和老师提问。

2、多人搜索的吃豆人

下载项目代码后，你可以玩一下吃豆人的游戏。

```
python pacman.py
```

现在，在 multiAgents.py 中运行提供的反射吃豆人。

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

请注意，即使在简单的布局上，它的成绩也很差：

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

检查其代码（multiAgents.py），并确保你了解它在做什么。

3、项目内容

问题 1（4 分）：反射吃豆人

改进 multiAgents.py 里的 ReflexAgent 。原有的 ReflexAgent 代码提供了一些有用的示例，这些方法用于查询 GameState 以获取信息。一个有能力的反射 Agent 必须同时考虑食物位置和鬼魂位置才能做得更好。你的 Agent 应该轻松可靠地清除 testClassic 地图：

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

在默认的 mediumClassic 布局上尝试你的反射 Agent，采用一个或两个鬼魂（你可以关闭动画以加快显示速度）：

```
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？它可能会经常在有 2 个幽灵的游戏板上死掉，除非你的评估函数很好。

注意：请记住，newFood 有 asList() 函数

注意：请尝试用重要值（例如到食物的距离）的倒数来做特征量，而不仅仅是该值本身。

注意：你编写的评估函数会评估状态——操作（state-action）对；在后面的部分中，你将评估状态本身（states）。

注意：你可能会发现查看各种目标对象（object）的内容可以帮助你的调试。你可以通过打印目标对象的字符串表示形式来查看其内容。例如，你可以使用 print（newGhostStates）来 打印 newGhostStates。

选项：默认的幽灵是随机的；你还可以通过-g DirectionalGhost 来使用稍微聪明些的定向幽灵。如果随机性让你无法判断你的 Agent 是否在改进，你可以使用 -f 的选项来固定的随机种子（这样，每个游戏都有相同的随机选择）。你还可以使用 -n 连续玩多个游戏。使用 -q 关闭图形以快速运行大量游戏。

评分：我们将在 openClassic 布局上运行你的 Agent 10 次。如果你的 Agent 超时或从未获胜，你将获得 0 分。如果你的 Agent 至少赢了 5 次，你将获得 1 分，如果你的 Agent 赢得所有 10 场比赛，你将获得 2 分。如果你的平均分数大于 500，你将获得额外的 1 分，如果大于 1000，你将额外获得 2 分。你可以在这些条件下测试你的 agent

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

假如你想要在没有图形的情况下运行，请使用：

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

请不要在这个问题上花太多时间，因为项目的主要部分还在后面。

问题 2（5 分）：最小最大值

现在，你将在 multiAgents.py 提供的 MinimaxAgent 类中编写一个对抗性搜索吃豆人。你的 minimax 吃豆人应该可以处理任意数量的幽灵，所以你必须编写一个比你之前在课程中看到的稍微通用些的算法。换句话说，你的最小最大树需要为每个最大层提供多个最小层（每个鬼魂一个）。

你的代码还应将游戏树扩展到任意深度。使用提供的 self.evaluationFunction 对最小最大值树的叶子进行评分，该函数默认为 scoreEvaluationFunction。MinimaxAgent 扩展（extend）了 MultiAgentSearchAgent，所以可以访问到 self.depth 和 self.evaluationFunction。确保 minimax 代码在适当的情况下会用到这两个变量，因为这些变量会根据命令行选项而赋值。

重要提示：单个的搜索移动是定义为一次吃豆人的移动和所有幽灵作出的反应，因此深度 2 的搜索是吃豆人和每个幽灵移动两次。

评分：我们将检查你的代码，以确定它是否探索了正确数量的游戏状态。这是检测 minimax 实现中一些细微错误的唯一可靠方法。因此，自动评分器对你调用 GameState.generateSuccessor 的次数会非常严厉。如果你调用它超过或少于必要的次数，自动评分器会抱怨。若要测试和调试代码，请运行

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

这将显示你的算法在许多小树上运行的结果，以及一个吃豆人游戏。假如你要在没有图形的情况下运行它，请使用：

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

一些提示和观察：

- 使用帮助函数来实现递归算法。
- 最小最大值的正确实现将导致吃豆人在某些测试中输掉游戏。这不是问题：因为它是正确的行为，它将通过测试。
- 这部分 Pacman 测试的评估函数已经写好了（self.evaluationFunction）。你不应该改变这个函数，但要认识到现在我们正在评估状态（state）而不是操作（action），可以和之前我们对反射 agent 所做的作比较。在这里我们评估未来状态，而反射 Agent 只会评估基于当前状态的操作。
- 对于深度 1、2、3 和 4，minimaxClassic 中初始状态的最小最大值分别为 9、8、7、-492。请注意，你的最小最大值 Agent 通常会获胜（ 665/1000 场比赛，基于我们的代码），尽管深度为 4 的最小最大值预测很糟糕。

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

- 吃豆人始终是 agent 0，agent 按索引（agent index）增加的顺序移动。
- minimax 中的所有状态都应该是 GameState，要么通过 GameState.generateSuccessor 生成，要么传递给 getAction。在此项目中，你不用抽象到简化状态。
- 在较大的地图上，如 openClassic 和 mediumClassic（默认），你会发现吃豆人擅长不死，但很不擅长获胜。他经常会四处乱窜，却没有进展。他甚至可能会在一个点旁边乱窜而不吃它，这是因为他不知道吃完那个点后该去哪里。如果你看到这种行为，请不要担心，问题 5 将清理所有这些问题。
- 当吃豆人认为自己的死亡是不可避免的时，他会因为生存下来的惩罚而努力尽快结束游戏。有时，这是对随机鬼魂的错误处理，但最小最大值 agent 总是假设最坏的情况。确保你明白为什么吃豆人在这种情况下会冲向最近的幽灵：

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

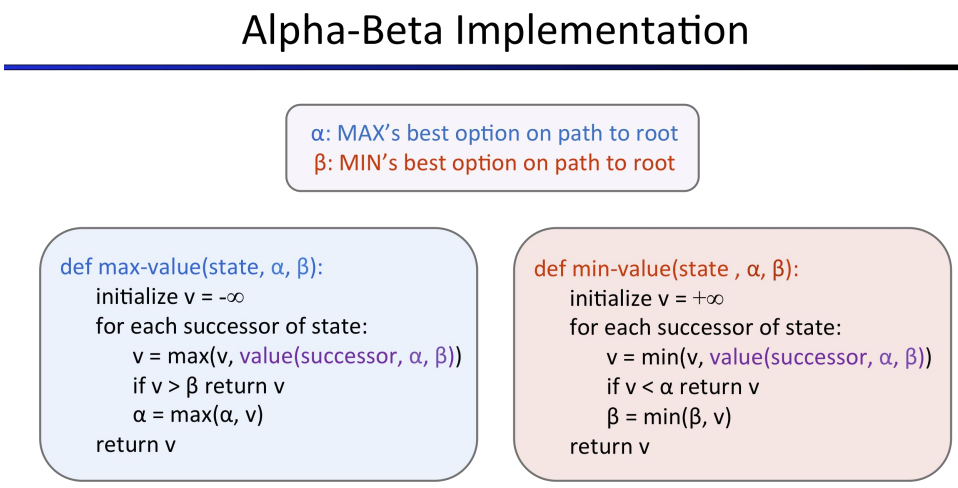
问题 3（5 分）：Alpha-Beta 修剪

创建一个新的 Agent，该 Agent 使用 alpha-beta 修剪来更有效地探索 AlphaBetaAgent 中的最小最大值树。同样，你的算法将比课堂中的伪代码稍微通用一些，因此部分的挑战是将 alpha-beta 修剪逻辑适当地扩展到多个最小化 agent。

你应该会看到加速（也许深度 3 的 alpha-beta 运行速度与深度 2 的最小最大值一样快）。理想情况下，smallClassic 上的深度 3 应该在每次移动几秒钟或更快的情况下运行。

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

AlphaBetaAgent 的最小最大值应与 MinimaxAgent 的最小最大值相同，尽管它选择的操作可能因不同的打破平局（tie-break）行为而有所不同。同样，对于深度 1、2、3 和 4，minimaxClassic 中初始状态的最小最大值分别为 9、8、7 和 -492。



图一、Alpha-Beta 算法的伪码

评分：我们会检查你的代码以确定它是否探索了正确的状态数，因此执行 alpha-beta 修剪时，你一定不能对子节点（children）重新排序。换句话说，后续状态（successor state）应该始终按

照 GameState.getLegalActions 返回的顺序进行处理。同样，不要对 GameState.generateSuccessor 做不必要的调用。

你一定不要为了匹配自动评分器探索的状态集而修剪相等状态。（确实，你可能会在尝试相等的情况下做修剪，并且对根的每个子节点调用一次 alpha-beta，但这与自动评分器不兼容。）

下面的伪码是你应该为此问题实现的算法。

若要测试和调试代码，请运行

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

这将显示你的算法在许多小树上运行，以及一个吃豆人游戏的结果。假如你要在没有图形的情况下运行它，请使用：

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

正确实施 alpha-beta 修剪将导致 Pacman 输掉一些测试。这不是问题：因为它是正确的行为，它将通过测试。

问题 4（5 分）：期待最大值

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战。任何曾经赢得井字游戏的人都可以告诉你，情况并非总是如此。在本问题中，你将实现 ExpectimaxAgent，这对于建模可能做出次优选择的 Agent 的随机行为非常有用。

与此类中尚未涵盖的搜索和问题一样，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速度，我们提供了一些基于泛型树的测试用例。你可以使用以下命令在小型游戏树上调试你的实现：

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

建议你对这些小型且可管理的测试用例进行调试，这将帮助你快速找到错误。

一旦你的算法在小树上工作，你就可以在 Pacman 中观察到它的成功。随机鬼魂当然不是最佳的最小最大值 agent，因此使用最小最大值搜索对它们进行建模可能不合适。ExpectimaxAgent 将不再对所有幽灵操作进行最小值操作，而是根据 agent 对幽灵行为方式的模型进行预测。为了简化代码，你可以假设你只会与随机选择其 getLegalActions 的对手运行。

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

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

你现在应该在 angent 与鬼魂近距离接触时观察到一种更漫不经心的行为。特别是，如果吃豆人意识到他可能被困住了，但可能会逃脱以多抓几块食物，他至少会尝试着做一下。检查以下两种方案的结果：

```
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
```

你应该发现你的 ExpectimaxAgent 赢了大约一半的时间，而你的 AlphaBetaAgent 总是输。确保你了解为什么此处的行为与最小最大值情况不同。

期望 expectimax 的正确实现将导致 Pacman 失去一些测试。这不是问题：因为它是正确的行为，它将通过测试。

问题 5（5 分）：评估函数

在提供的函数 betterEvaluationFunction 中为 Pacman 编写一个更好的评估函数。评估函数应评估状态（state），而不是像反射 agent 评估函数那样评估操作（action）。使用深度 2 搜索时，你的评估函数应该在一半以上的时间内清除一个随机鬼魂的 smallClassic 布局，并且仍然以合理的速度运行（为了获得全部学分，吃豆人获胜时的平均得分应该在 1000 分左右）。

评分：自动评分器将在 smallClassic 布局上运行你的 agent 10 次。我们将通过以下方式为你的评估函数分配分数：

- 如果你至少赢了一次而没有使自动评分器超时，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
- 平均得分至少为 500 分为 +1，平均分至少为 1000 分为 +2（包括输掉比赛的分数）
- +1 如果你的游戏在自动评分机上平均花费的时间少于 30 秒，使用 --no-graphics 运行时。
- 平均分数和计算时间的额外分数只有在你至少赢得 5 次时才会获得。

你可以在这些条件下试用你的 Agent

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

在没有图形的情况下运行它，请使用：

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

4、项目报告

简要清晰地描述完成项目时遇到的困难，采用的解决方法，提出改进意见，和总结每个小组成员的贡献。

5、提交

在提交你的解答之前，你需要通过执行 submission_autograder.pyc 来产生几个文件。在运行这个程序之前，你必须确认所有与 autograde 有关的文件都处在原始状态，没有做过任何的改动。假如你编辑过任何 autograde 的文件，请重新下载一份项目代码，仅仅替换你作解答的文件，否则运行 submission_autograder.pyc 将无法通过。

此外，submission_autograder.pyc 要在 Python 3.6（准确的说是 3.6.13，你可以用 Anaconda 来安装正确的 Python 版本）下执行，否则会报错。

最后，submission_autograder.pyc 需要用 rsa 库来给你的成绩加密，假如你没有的话，请用下面的命令安装 rsa 库。

```
conda install -c conda-forge rsa
```

或者用下面的命令，假如你没有 conda。

```
pip install rsa
```

进到你的 multiagent 文件夹里，执行以下命令：

```
python submission_autograder.pyc
```

成功执行后，该命令会输出你的各个题目的得分和最后总分，并在 grade 文件夹里会生成一个 log 文件和一个 token 文件。确认该分数和你自己运行 autograder.py 得到的分数相同后，将整个 grade 文件夹和你修改过的文件（其它没有修改过的，例如 autograder.py，不需要，但假如你用到了之前项目 1 的文件，请将它们也一道打包），以及项目报告，放在一个以你的组名命名的文件夹里，生成一个 zip 文件，一并提交上来。