人工智能基础 LAB1 实验要求

DDL：2021.5.28 23:59:59

一、评分标准

项目内容	分值
BFS	2
A*	2
minimax	2
alpha-beta	2
实验报告	2
迟交或不符合格式	倒扣分

二、实验内容与提示

整体描述

本次实验有 2 个部分，分别是 Search 和 Multiagent。具体而言，Search 的目标是吃豆人仅仅是寻找食物；Multiagent 的目标是吃完所有食物，同时避开鬼。抽象而言，Search 实现的静态查找算法，Multiagent 的问题是在有对手的情况下做出下一步决策使自己的利益最大化。

Search 部分需要你实现 BFS 算法和 A*算法。Multiagent 部分需要你实现 minimax 算法和 alpha-beta 剪枝。**你只需要并且只能修改并向助教提交 myImpl.py 文件**，阅读其他代码对完成实验没有意义。请不要在 myImpl.py 文件中 import 其他模块，否则会造成测试失败。实验代码量大约为 100 行以内。

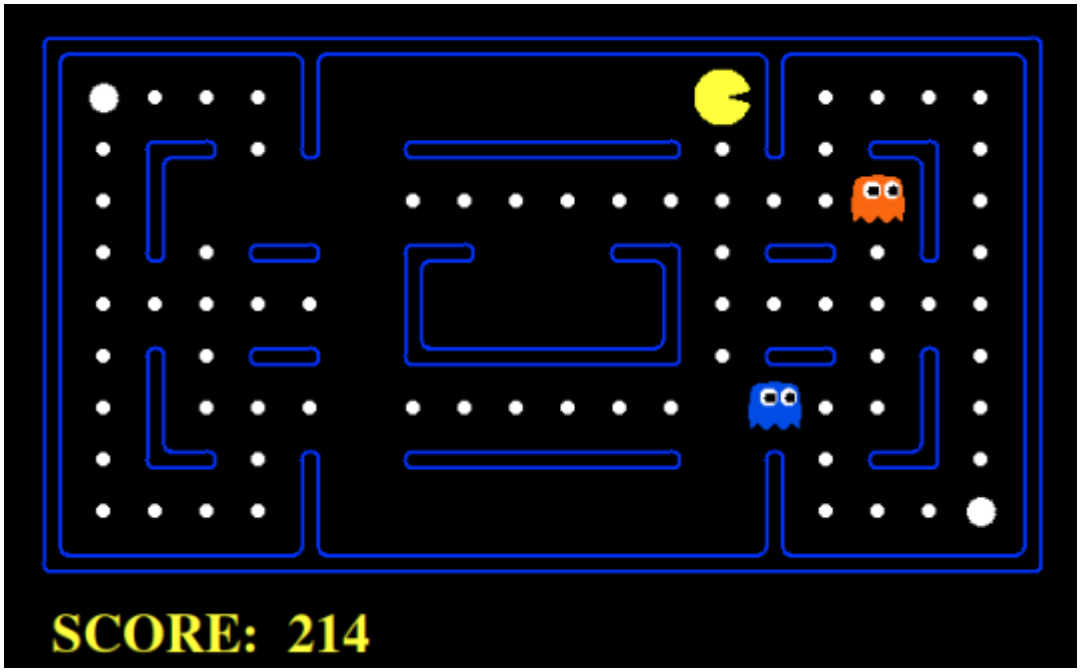
实验需要使用 Python 3.6 版本，建议使用 anaconda 来管理 Python 环境。本实验推荐使用 Linux，测试只需要在命令行中运行 ./test.sh。正确代码应该 PASS 所有的测试。如果你实现的代码有误，请善用报错信息和 print()函数。

```
Question q3
===========

*** PASS: test_cases/q3/0-eval-function-lose-states-1.test
*** PASS: test_cases/q3/0-eval-function-lose-states-2.test
*** PASS: test_cases/q3/0-eval-function-win-states-1.test
*** PASS: test_cases/q3/0-eval-function-win-states-2.test
*** PASS: test_cases/q3/0-lecture-6-tree.test
*** PASS: test_cases/q3/0-small-tree.test
*** PASS: test_cases/q3/1-1-minmax.test
*** PASS: test_cases/q3/1-2-minmax.test
*** PASS: test_cases/q3/1-3-minmax.test
*** PASS: test_cases/q3/1-4-minmax.test
*** PASS: test_cases/q3/1-5-minmax.test
*** PASS: test_cases/q3/1-6-minmax.test
*** PASS: test_cases/q3/1-7-minmax.test
*** PASS: test_cases/q3/1-8-minmax.test
*** PASS: test_cases/q3/2-1a-vary-depth.test
*** PASS: test_cases/q3/2-1b-vary-depth.test
*** PASS: test_cases/q3/2-2a-vary-depth.test
*** PASS: test_cases/q3/2-2b-vary-depth.test
*** PASS: test_cases/q3/2-3a-vary-depth.test
*** PASS: test_cases/q3/2-3b-vary-depth.test
*** PASS: test_cases/q3/2-4a-vary-depth.test
*** PASS: test_cases/q3/2-4b-vary-depth.test
*** PASS: test_cases/q3/2-one-ghost-3level.test
*** PASS: test_cases/q3/3-one-ghost-4level.test
*** PASS: test_cases/q3/4-two-ghosts-3level.test
*** PASS: test_cases/q3/5-two-ghosts-4level.test
*** PASS: test_cases/q3/6-tied-root.test
*** PASS: test_cases/q3/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2c-check-depth-two-ghosts.test
```

如果你想更好的了解游戏规则，体验一下实验的乐趣，可以先玩一局吃豆人。在命令行中输入以下命令即可。

```
cd search
python pacman.py
```



附：Conda 环境创建

```
conda create --name ustc-ai python=3.6
source activate ustc-ai
```

Search

你需要实现 BFS 算法和 A*算法。你只需要填写 myBreadthFirstSearch 和 myAStarSearch 两个函数。函数的返回值为从初始状态到目标状态所经过的所有状态的列表。**实现时请删去 util.raiseNotDefined()**。

```
def myBreadthFirstSearch(problem):
    # YOUR CODE HERE
    util.raiseNotDefined()
    return []

def myAStarSearch(problem, heuristic):
    # YOUR CODE HERE
    util.raiseNotDefined()
    return []
```

函数的参数 problem 可以调用 3 个函数：

- 函数 getStartState 可以获得该 problem 的初始状态。

```
start_state = problem.getStartState()
```

- 函数 isGoalState 可以判断当前状态 state 是否为目标状态。

```
problem.isGoalState(state) == True
```

- 函数 getChildren 可以获得 state 后可以到达的一系列状态。返回值是由二元组(next_state, step_cost)组成的列表。next_state 是下一状态，step_cost 是从 state 到 next_state 需要的代价。

```
children = problem.getChildren(state)
```

参数 heuristic 本身就是一个函数，可以获得当前状态到目标状态的启发式估计值。

```
h_n = heuristic(state)
```

我们已经给出了 DFS 函数的参考代码，请仔细阅读并参考。

```
def myDepthFirstSearch(problem):
```

你可能还需要使用我们提供的栈、队列和优先队列这些数据结构。

大家在学习数据结构时可能都已经熟悉了栈和队列。它们的特点可以分别简单概括为先进后出和先进先出。

```
stack = util.Stack()
stack.push('eat')
stack.push('study')
stack.push('sleep')
stack.pop() == 'sleep'

queue = util.Queue()
```

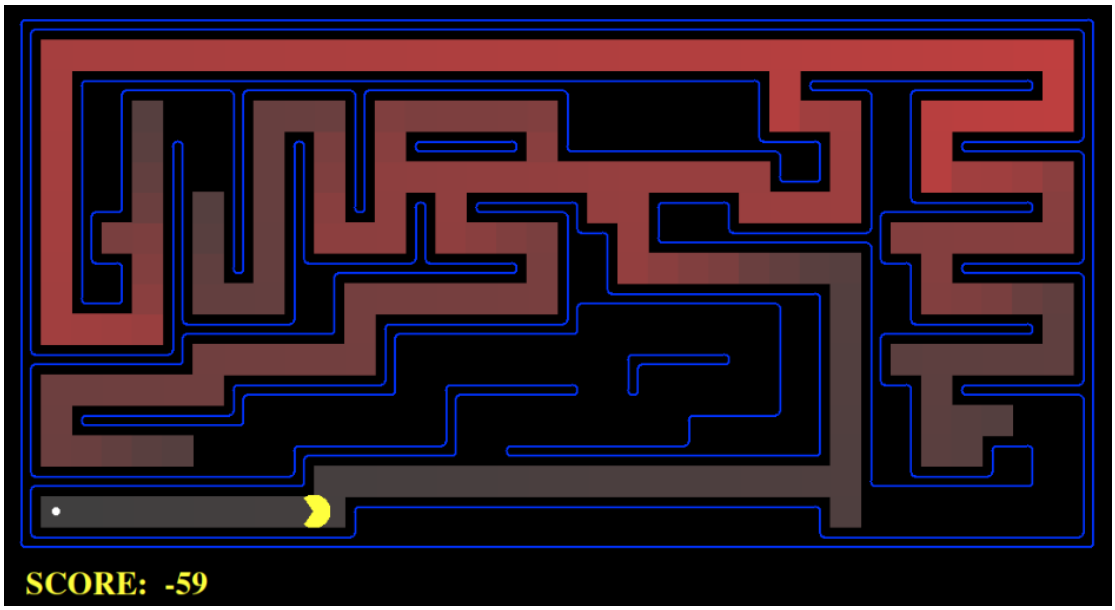
```
queue.push('eat')
queue.push('study')
queue.push('sleep')
queue.pop() == 'eat'
```

优先队列的使用则需要赋予一个表示优先性的值，值越小就会越先出队。

```
pq = util.PriorityQueue()
pq.update('eat', 2)
pq.update('study', 1)
pq.update('sleep', 3)
pq.pop() == 'study'
```

3 个数据结构都有 isEmpty 函数来判断数据结构内部是否有数据。

最终测试时，会动画显示 3 种搜索方法选择的路径以及搜索过的状态（红色表示），请比较一下三者的区别。

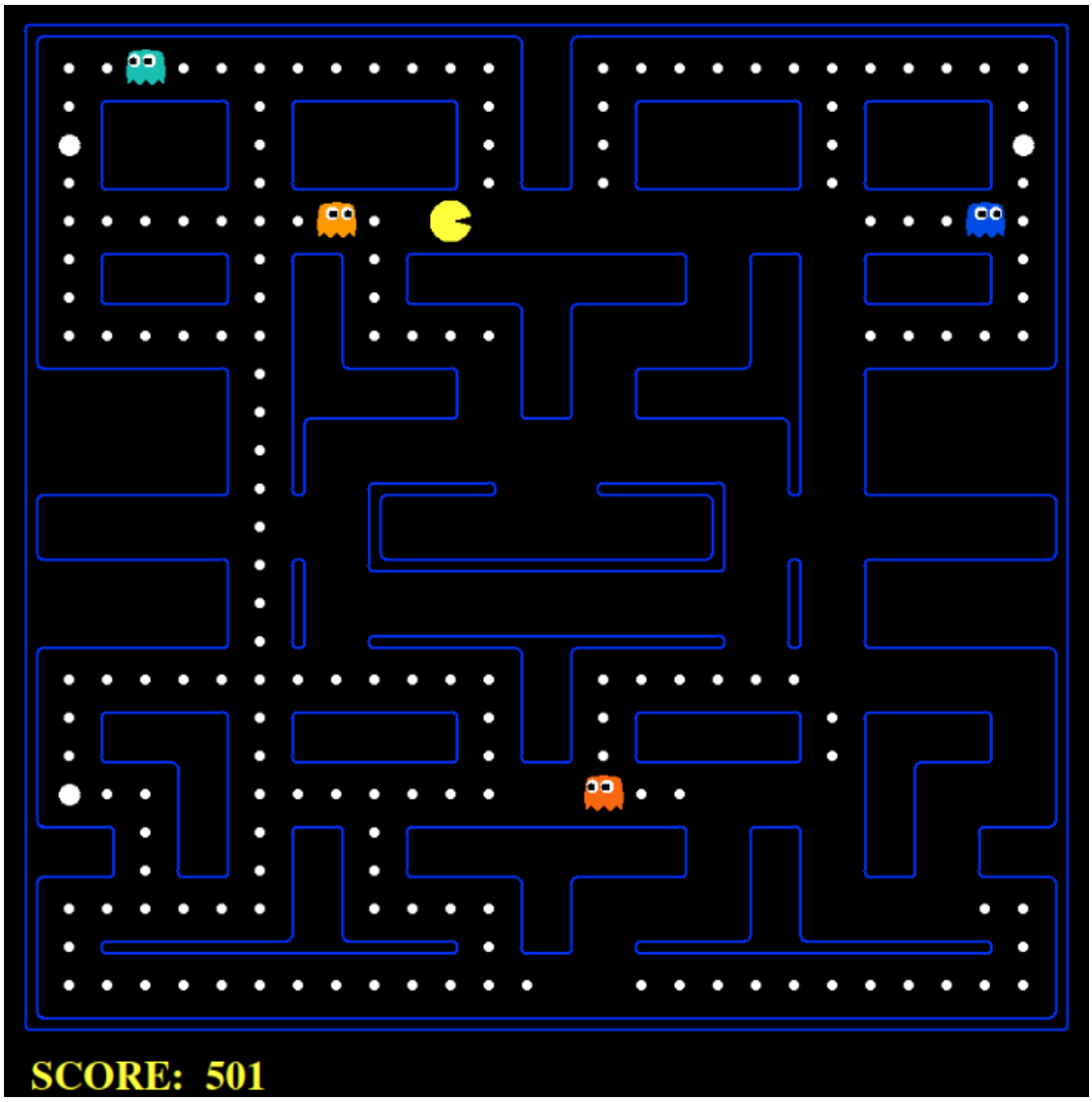


MultiAgent

Multiagent 的问题是在有对手的情况下做出下一步决策使自己的利益最大化。游戏中目标 agent 为吃豆人，其他 agent 为鬼。

我们已经实现了一个只基于当前状态做出反应的吃豆人，你可以输入以下命令查看它的表现。目录 multiagent/layouts 中有不同的游戏场景，你可以更改-l 后的选项。你可以更改-p 后的选项为 MinimaxAgent 或 AlphaBetaAgent 来测试你实现的算法。

```
cd multiagent
python pacman.py -p ReflexAgent -l originalClassic --frameTime 0
```



你需要实现 minimax 算法和 alpha-beta 剪枝则会提前预估几步，在最坏的打算下最优化自己的效用。你只需要**填写 MyMinimaxAgent 和 MyAlphaBetaAgent 两个类**。其中函数 getNextState 会被外部程序调用，获得当前状态下最优的下一个状态。你可能需要添加一些辅助函数来进行递归调用。

参数 state 可以调用 4 个函数。

- 函数 isTerminate 将返回当前状态 state 是否已经停止。停止状态意味着不会有下一个状态，游戏中指已经赢了或输了。

```
state.isTerminated() == True
```

- 函数 isMe 将返回是否为目标 agent 在进行操作。在游戏中，True 表示轮到吃豆人采取

移动操作，False 表示轮到某个鬼在采取移动操作。你可以用来判断当前应该最大化还是最小化效用。

```
state.isMe() == True
```

- 函数 getChildren 将返回当前状态 state 接下来所有可能的状态。请使用 for 来遍历。注意：alpha-beta 剪枝的目的是缩小搜索空间，如果在遍历 getChildren() 中，MyAlphaBetaAgent 发现可以剪枝，请停止遍历。

```
for child in state.getChildren():
```

- 函数 evaluateScore 将返回当前状态对于目标 agent 的效用。

```
score = state.evaluateScore()
```

值得注意的是算法搜索的深度 depth，它指的是每个 agent 所走的步数。例如 depth=2，有 1 个 pacman 和 2 个 ghost，则从搜索树的最顶层到最底层应该经过 pacman->ghost1->ghost2->pacman->ghost1->ghost2，操作应该为 max->min->min->max->min->min。

三、提交

DDL：2021 年 5 月 28 日 23:59:59。逾期扣分。

提交方式：bb 系统作业区。

提交内容（格式）：

```
LAB1 _PB18000001__张三\
        |--- report.pdf
        |--- myImpl.py
```

注意事项：

（1） 文件命名方式需按上述要求，不得命名为“LAB1_张三_PB18000001”等；

（2） 实验报告**请提交 pdf 格式**，不接受 doc、docx、md、tex 等格式；

（3） 代码**请提交且只能提交 myImpl.py，不得提交其余任何代码文件**（包括但不限于 sh、py 等）【所以你只能对 myImpl.py 进行修改，不得修改其他文件】

四、附件

链接：https://rec.ustc.edu.cn/share/71531310-a44f-11eb-8fd2-a7ccc52c33b4