

# Artificial Intelligence

## Assignment 4: Multi-Agent Search

**Dahuin Jung**

School of Computer Science and Engineering

Soongsil University

2024



**Soongsil University**

# Table of Contents

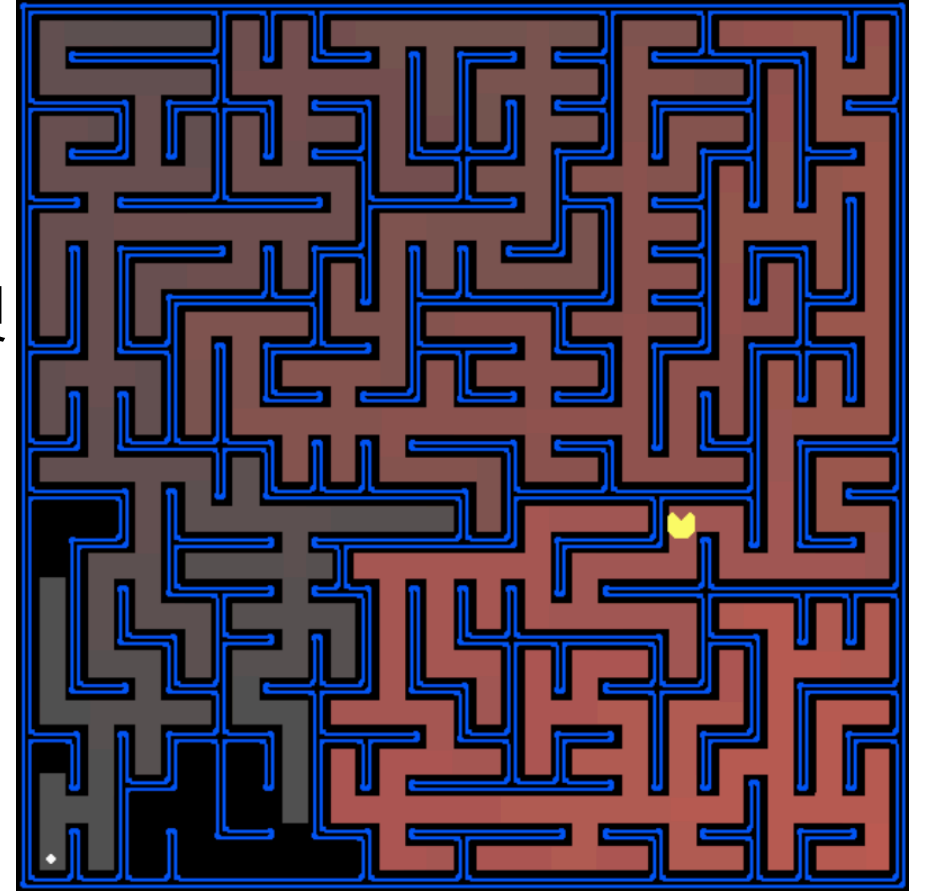
- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

# Introduction

- 이 project에서 classic version의 Pacman agent를 설계할 것이며 이 과정에서 유령을 포함하고 minimax search를 구현할 것입니다.



# Files

- 이 project code는 여러 개의 Python file들로 구성되어 있으며, 과제를 완료하기 위해 참고만 하는 file들이 있고, 무시하는 file들(수정 x), 수정해서 제출하는 file이 있습니다.

수정해야할 file:	
<code>multiAgents.py</code>	multi-agent search agent들이 있을 file

### 참고하는 파일들:

<code>pacman.py</code>	Pacman Game을 실행하는 주요 file. 이 file은 이 project에서 사용하게 될 PacmanGameState 타입을 설명
<code>game.py</code>	Pacman 세계가 작동하는 논리를 담고있는 file이며 이 파일은 AgentState, Agent, Direction Grid와 같은 여러 지원 타입들을 설명
<code>util.py</code>	search algorithm 구현에 유용한 데이터 구조를 포함한 file

### Supporting files you can ignore:

<code>graphicsDisplay.py</code>	Pacman을 위한 그래픽
<code>graphicsUtils.py</code>	Pacman graphic을 보조해주는 file
<code>textDisplay.py</code>	Pacman을 위한 ASCII 그래픽 file
<code>ghostAgents.py</code>	유령을 제어하는 Agent file

## Supporting files you can ignore: (cont'd)

<code>keyboardAgents.py</code>	Pacman을 제어하기 위한 키보드 인터페이스 file
<code>layout.py</code>	Layout file을 읽고 해당 내용을 저장하는 code를 담은 file
<code>autograder.py</code>	Project 의 자동 채점기
<code>testParser.py</code>	자동 채점기의 test 및 solution file을 parsing하는 file
<code>testClasses.py</code>	일반적인 자동 채점 test class를 포함한 file
<code>test_cases/</code>	각 문제에 대한 test case가 포함된 directory
<code>searchTestClasses.py</code>	Assignment 4에 특화된 자동 채점 test class가 포함된 file

# Files

- 수정하고 제출해야 될 file:
  - 과제를 진행하면서 `multiAgents.py` 를 수정하고 이 file만 제출하면 됩니다.
  - 해당 file만 LMS 과제 제출 section에 upload 해주세요.

수정해야할 file:	
<code>multiAgents.py</code>	multi-agent search agent들이 있을 file




# Autograding

- 이 명령은

`python autograder.py`

3가지 문제에 대한 여러분의 solution을 채점합니다.

- File을 편집하기 전 실행하면 한두page정도의 출력 결과를 확인할 수 있습니다. 
- 각 solution의 구현을 완료한 후에는 `raiseNotDefine()` 함수를 제거하시면 됩니다.

```
Question q1
=====
Pacman died! Score: -749
Pacman died! Score: 28
Pacman died! Score: -403
Pacman died! Score: -58
Pacman died! Score: -226
Pacman died! Score: -364
Pacman emerges victorious! Score: 479
Pacman died! Score: -82
Pacman died! Score: -203
Pacman died! Score: -302
Average Score: -188.0
Scores:      -749.0, 28.0, -403.0, -58.0, -226.0, -364.0, 479.0, -82.0, -203.0, -302.0
Win Rate:    1/10 (0.10)
Record:      Loss, Loss, Loss, Loss, Loss, Loss, Win, Loss, Loss, Loss
*** FAIL: test_cases/q1/grade-agent.test (0 of 4 points)
***      -188.0 average score (0 of 2 points)
***      Grading scheme:
***          < 500: 0 points
***          >= 500: 1 points
***          >= 1000: 2 points
***      10 games not timed out (0 of 0 points)
***      Grading scheme:
***          < 10: fail
***          >= 10: 0 points
***      1 wins (0 of 2 points)
***      Grading scheme:
***          < 1: fail
***          >= 1: 0 points
***          >= 5: 1 points
***          >= 10: 2 points

### Question q1: 0/4 ###

Question q2
=====
*** Method not implemented: getAction at line 138 of multiAgents.py
*** FAIL: Terminated with a string exception.

### Question q2: 0/5 ###

Question q3
=====
*** Method not implemented: getAction at line 150 of multiAgents.py
*** FAIL: Terminated with a string exception.

### Question q3: 0/5 ###
```

# Autograding

- 4가지 질문 각각에 대해, 이 명령은 해당 질문의 test 결과, 문제 별 점수, 그리고 마지막에는 전체 요약 을 출력합니다.
- 아직 문제들을 해결하지 못했기 때문에 모든 test가 실패할 것입니다.
  - 각 문제들을 해결하다 보면 일부 test는 통과하고 다른 test는 실패할 수 있습니다.
  - 모든 test를 통과하시면 해당 문제에서 만점을 받으실 수 있습니다.

# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

# Setup

- 이번 과제에서는 GPU가 필요하지 않기 때문에 각자 자신의 컴퓨터나 노트북에서 과제를 진행하시면 됩니다(환경 설정은 이후 local setup에서 참고할 수 있습니다).
  - code를 실행하기 전에 다음 명령어를 실행하세요: `conda activate AI-24`

```
(base) C:\Users\ssu_hai>conda activate AI-24  
(AI-24) C:\Users\ssu_hai>
```

# Local setup

- Step 1 Anaconda download

- <https://www.anaconda.com/download/success> (download)

- Step 2 Anaconda prompt에 접속 후 AI-24를 활성화

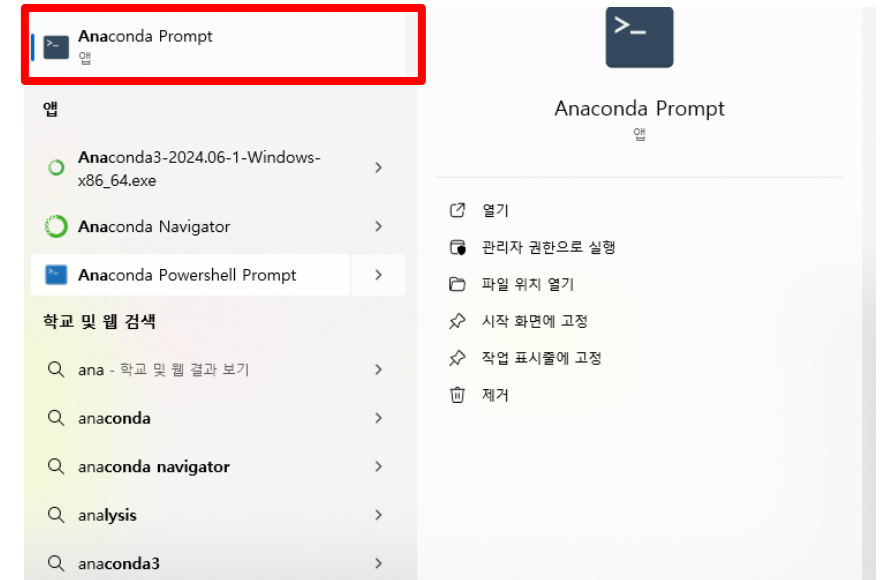
- 1. `conda env create -f environment.yml`(`cd` 명령어를 사용해서 `env` 폴더로 이동 후, 다음 명령어를 실행하세요.)

```
(base) C:\Users\ssu_hai\Desktop\인공지능\Assignment0>cd env
```

- 2. `(base) C:\Users\ssu_hai\Desktop\인공지능\Assignment0\env>conda env create -f environment.yml`

```
(base) C:\Users\ssu_hai>conda activate AI-24
```

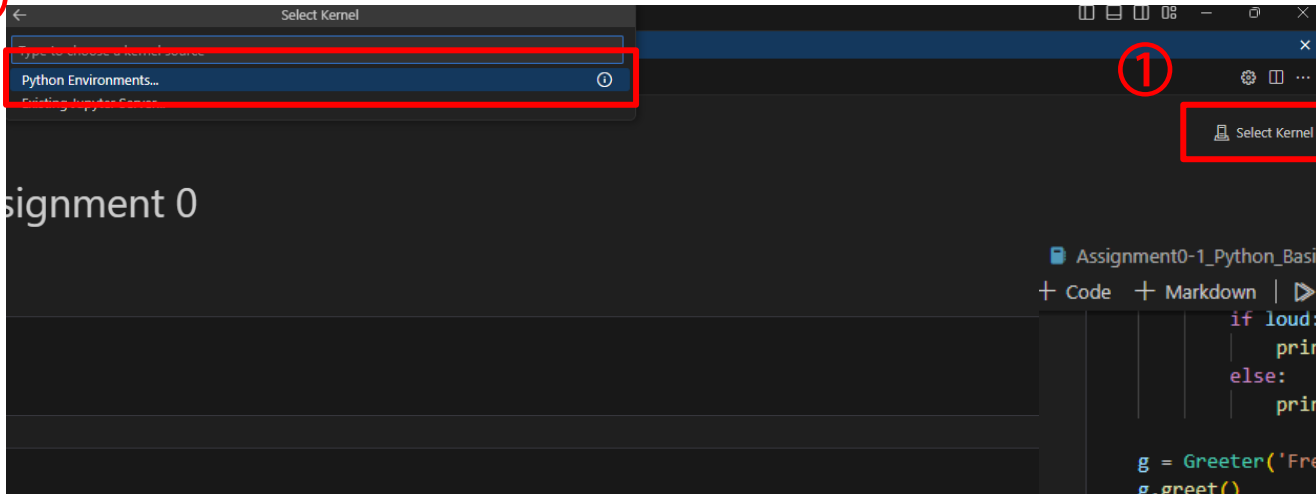
```
(AI-24) C:\Users\ssu_hai>
```



# Local setup

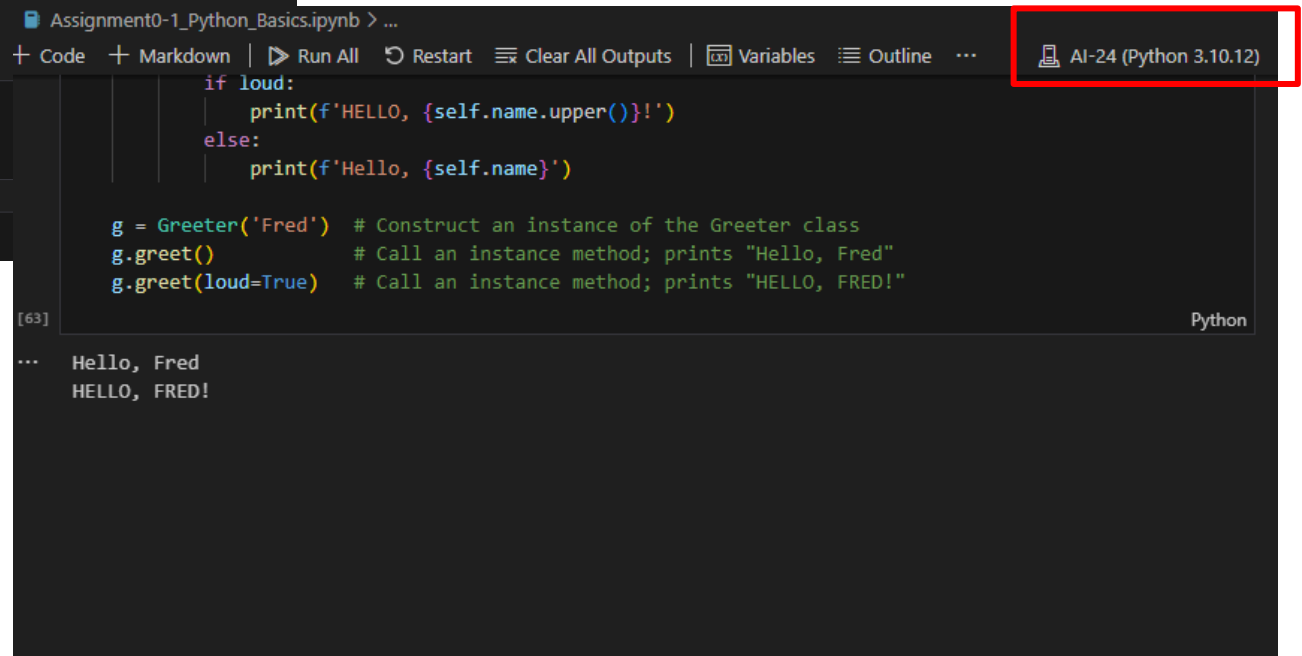
- Step 3 활성화 확인

②



- steps 1부터 step 3 까지 완료  
하시면 code를 실행할 수 있  
습니다!

③



# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

# Welcome to Multi-Agent Pacman

- code file(`AS4_multiagentsearch.zip`) 를 download하여 압축을 풀고 해당 directory로 이동한 후, 아래 명령어를 실행하면 classic Pacman game을 할 수 있습니다. `python pacman.py`
- 그리고 방향키를 사용해서 이동할 수 있습니다.
- 이제 `multiAgents.py` 에 있는 제공된 `ReflexAgent`를 실행하세요:  
`python pacman.py -p ReflexAgent`



# Welcome to Multi-Agent Pacman

- 참고로 이 Agent는 간단한 layout에서도 성능이 좋지 않습니다.

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

- `multiAgents.py`에 있는 code를 검토하고 ReflexAgent가 어떻게 하는지 확인해보세요.

# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

# Q1: Reflex Agent

- `multiAgents.py`에 있는 `ReflexAgent`를 개선하여 제대로 play할 수 있도록 만드시면 됩니다. `GameState` 에서 정보를 조회하는 몇가지 유용한 method 예시가 있습니다. 성능이 좋은 `ReflexAgent`는 음식 위치와 유령 위치를 모두 고려해야 합니다. 여러분의 에이전트는 `testClassic` layout을 쉽게 안정적으로 clear할 수 있어야 합니다:

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

# Q1: Reflex Agent

- 여러분의 ReflexAgent를 기본 mediumClassic layout에서 한 명 또는 두 명의 유령과 함께 실행해보세요(animation을 끄면 출력 속도가 빨라집니다):

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

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

- 여러분의 agent는 어떻게 작동하나요? 기본 board에서 2명의 유령과 함께 할 경우, 평가 함수가 좋지 않으면 자주 죽게 될 것입니다.

# Q1: Reflex Agent

- 채점 방식: openClassic layout에서 Agent를 10번 실행할 예정입니다. Agent가 시간 초과되거나 한 번도 승리하지 못하면 0점을 받습니다.
- Agent가 최소 5번 승리하면 1점을, 10번 모두 승리하면 2점을 받습니다.
- 평균 점수가 500을 초과하면 추가로 1점, 1000을 초과하면 2점을 추가로 받습니다.
- 이 조건에서 Agent를 test하려면 아래 명령어를 사용하세요:

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

- Graphic없이 실행하려면 아래를 사용하시면 됩니다:

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

# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

## Q2: Minimax

- 이제 `multiAgents.py`에 제공된 `MinimaxAgent` class에 적대적 탐색 agent를 작성할 것입니다.
- 여러분의 minimax agent는 여러 유령이 있는 경우에도 작동해야 하므로, 강의에서 배운 것보다 약간 더 일반적인 알고리즘을 작성해야 합니다.
- 특히, minimax tree는 각 max layer마다 여러 개의 min layer(유령당 하나씩)를 갖게 됩니다.

## Q2: Minimax

- 여러분의 code는 game tree를 임의의 깊이까지 확장해야 합니다.
- Minimax tree의 leaf node는 제공된 `self.evaluationFunction`으로 평가하세요. 기본적으로 이 함수는 `scoreEvaluationFunction`으로 설정합니다.
- MinimaxAgent는 MultiAgentSearchAgent를 확장하며, 이를 통해 `self.depth`와 `self.evaluationFunction`에 접근할 수 있습니다.
- Minimax code를 작성할 때, 이 두 변수를 적절히 참조하도록 하세요. 이 변수들은 command line option에 따라 설정됩니다.



## Q2: Minimax

- 채점방식: 여러분의 code가 올바른 개수의 game 상태를 탐색하는지 확인할 것입니다. 이는 minimax 구현에서 발생할 수 있는 미묘한 bug를 감지하는 유일하게 신뢰할 수 있는 방법입니다. 따라서 자동 채점기는 GameState.generateSuccessor 호출 횟수에 매우 까다롭게 동작합니다. 필요한 것보다 더 많이 호출하거나 덜 호출하면 자동 채점기가 오류를 제기할 것입니다.

code를 test하고 debug하려면 아래 명령어를 실행하세요:

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

- 중요: 하나의 탐색 ply는 pacman의 한 번의 움직임과 모든 유령의 응답을 포함합니다. 따라서 depth 2의 탐색은 pacman과 각각의 유령이 두 번씩 움직이는 것을 의미합니다.

## Q2: Minimax

- 힌트:

- Minimax의 올바른 구현은 일부 test에서 pacman이 지는 경우가 나오는 것은 정상입니다. 이렇게 되면 test가 통과됩니다.
- 이번 단계의 pacman test를 위한 평가 함수(`self.evaluationFunction`)는 이미 작성되어 있습니다. 이 함수는 변경하지 않아야 하지만, 이번에는 현재 상태의 행동이 아닌, 상태를 평가한다는 점을 인지해야 합니다.
  - Look-ahead agent는 미래 상태를 평가합니다.
  - Reflex agent는 현재 상태에서 행동을 평가합니다.
- minimaxClassic layout의 초기 상태에 대한 minimax 값은 depth 1,2,3,4에서 각각 9, 8, 7, -492 입니다. Depth 4 minimax의 비관적인 예측에도 불구하고, 여러분의 minimax agent는 종종 승리할 것입니다. (이번 test에서는 1000번중 665번 승리)

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

# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission

## Q3:Alpha-Beta Pruning

- AlphaBetaAgent에서 alpha-beta 가지치기를 사용하여, minimax tree를 더 효율적으로 탐색하는 agent를 만드세요. 이번에도 강의에서 제공된 의사code보다 더 일반적인 알고리즘을 작성해야 하며, 여러 min agent에 alpha-beta 가지치기 logic을 적절히 확장하는 것이 목표의 일부입니다.
- Alpha-beta 가지치기를 사용하면 속도가 빨라지는 것을 확인할 수 있습니다(예시.depth 3의 alpha-beta 탐색이 depth 2의 minimax 탐색만큼 빠르게 실행될 수 있습니다). 이상적으로는, smallClassic에서 depth 3 탐색이 이동마다 몇 초 내에 실행되거나 그보다 더 빠르게 실행되어야 합니다.

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

## Q3:Alpha-Beta Pruning

- AlphaBetaAgent의 minimax 값은 MinimaxAgent의 minimax 값과 동일해야 하지만 동작 선택은 다른 동점 처리 방식 때문에 달라질 수 있습니다. 다시 말해,minimaxClassic layout에서 초기 상태의 값은 depths 1,2,3 4에서 각각 9,8,7,-492입니다.

## Q3:Alpha-Beta Pruning

- 채점 기준: code가 올바른 상태 수를 탐색하는지 확인하기 때문에, 자식 노드의 순서를 재정렬하지 않고 alpha-beta 가지치기를 수행하는 것이 중요합니다. 즉 후속 상태는 항상 `GameState.getLegalActions` 에 의해 반환된 순서대로 처리해야 합니다. 또한, `GameState.generateSuccessor`를 불필요하게 호출하지 않아야 합니다.
  - Alpha-beta 가지치기를 올바르게 구현하면 pacman이 일부 test에서 질 수 있습니다. 이는 정상적이며, 이렇게 되면 test를 통과합니다.
- 동등성에서 가지치기를 수행하지 않아야 자동 채점기가 탐색한 상태 집합과 일치합니다.

## Q3:Alpha-Beta Pruning

- Code를 test하고 debug하려면 아래 명령어를 실행하세요.

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

- 이 명령은 여러분의 알고리즘이 작은 트리들과 pacman game에서 어떻게 작동하는지 보여줍니다  
graphic없이 실행하려면 아래 명령어를 실행하세요:

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

# Table of Contents

- Introduction
- Setup
- Welcome to Multi-Agent Pacman
- Q1: Reflex Agent
- Q2: Minimax
- Q3: Alpha-Beta Pruning
- Submission



# Autograding

- 각 문제를 해결하면서 모든 test를 통과할 수 있습니다.
  - 문제에 대한 모든 test를 통과하면 만점을 받게 됩니다.

```
*** 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
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:      84.0
Win Rate:    0/1 (0.00)
Record:      Loss
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q3/8-pacman-game.test

### Question q3: 5/5 ###

Finished at 13:04:16

Provisional grades
=====
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
=====
Total: 14/14
```

# Files

- 수정하고 제출해야 될 file:
  - 과제를 진행하면서 `multiAgents.py` 를 수정하고 이 file만 제출하면 됩니다.
  - 해당 file만 LMS 과제 제출 section에 upload 해주세요.

수정해야할 file:	
<code>multiAgents.py</code>	multi-agent search agent들이 있을 file

# Submitting your work

- 제출하는 법
  - `multiAgents.py`
  - 이 file을 수정한 후 저장하고 바로 LMS에 제출하면 됩니다.

