

Artificial Intelligence

Assignment 3: Search

Dahuin Jung

School of Computer Science and Engineering

Soongsil University

2024



Soongsil University

Table of Contents

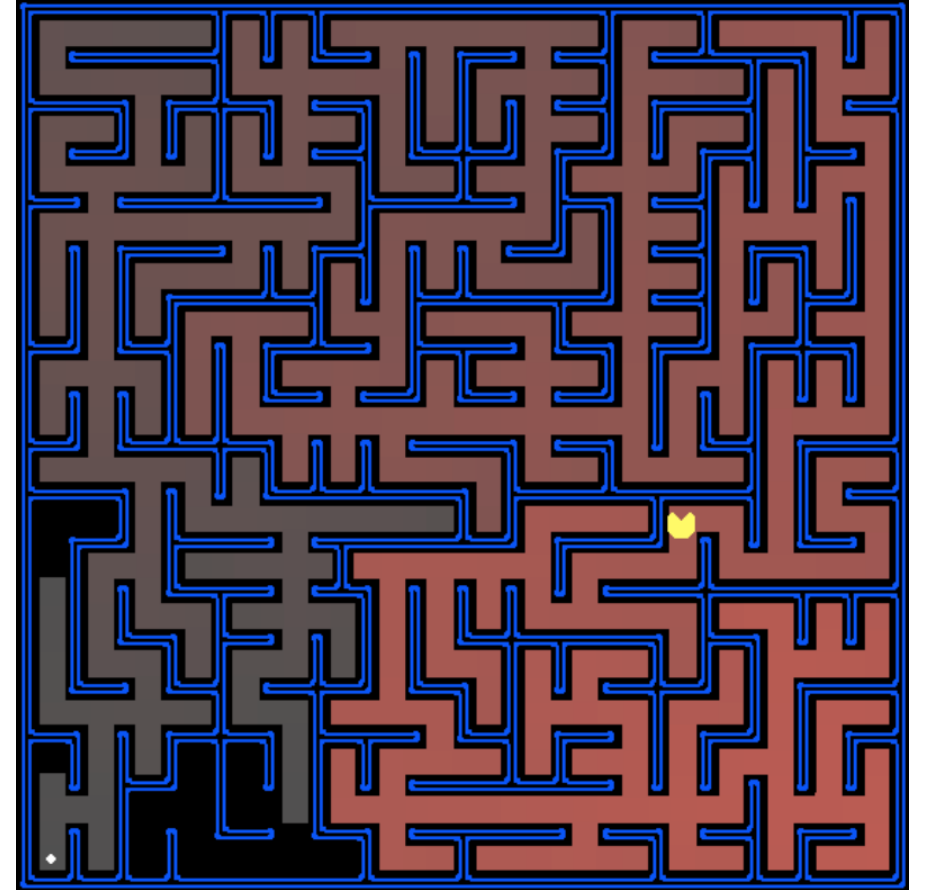
- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- Q4: A* Search
- Submission

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- Q4: A* Search
- Submission

Introduction

- 이 과제에서 **pacman 에이전트**를 통해 탐색 알고리즘을 만들어서 최적의 경로를 찾을 것입니다.



Files

- 이 과제에는 여러 개의 Python파일로 구성되어 있습니다.
- 이 파일중에서는 수정해야 될 파일, 읽어야할 파일(수정 x), 무시하면 되는 파일이 있습니다.

수정해야할 파일:	
search.py	모든 탐색 알고리즘이 들어가야할 곳

읽어봐야할 파일들(수정 x):

<code>pacman.py</code>	Pacman 게임을 실행하는 주요 파일로, gameState 타입이 포함되어 있어 프로젝트에서 사용
<code>game.py</code>	Pacman 게임의 로직을 담당하며, AgentState, Agent, Direction, Grid 와 같은 여러 지원 타입이 포함
<code>util.py</code>	탐색 알고리즘 구현에 유용한 데이터 구조들을 제공
<code>searchAgents.py</code>	모든 탐색 기반 에이전트가 포함될 파일 <code>goWest, goEast, ...</code>

무시하면 되는 파일들(수정 x):

<code>graphicsDisplay.py</code>	Pacman 그래픽 담당
<code>graphicsUtils.py</code>	Pacman 그래픽 지원을 제공
<code>textDisplay.py</code>	Pacman ASCII 그래픽 담당
<code>ghostAgents.py</code>	유령을 제어하는 에이전트를 담당

무시하면 되는 파일들(수정 x):

<code>keyboardAgents.py</code>	Pacman을 키보드로 제어하기 위한 인터페이스를 제공
<code>layout.py</code>	레이아웃 파일을 읽고 내용을 저장하는 코드
<code>autograder.py</code>	프로젝트의 자동 채점 도구
<code>testParser.py</code>	자동 채점 테스트 및 solution 파일을 파싱
<code>testClasses.py</code>	일반적인 자동 채점용 테스트 클래스들을 포함
<code>test_cases/</code>	각 질문에 대한 테스트 케이스가 포함된 디렉토리
<code>searchTestClasses.py</code>	과제 3의 특정 자동 채점 테스트 클래스

Files

- 수정 및 제출해야 하는 파일:

- `search.py`: 모든 탐색 알고리즘은 이 파일에 구현하여 과제 완료 후 이 파일을 제출하면 됩니다.
- 해당 파일만 LMS과제 제출란에 올려 주시면 됩니다.

Files you'll edit:	
<code>search.py</code>	모든 탐색 알고리즘이 들어가야할 곳

Autograding

- 명령어:

```
python autograder.py
```

명령어를 실행하면 작성한 4가지 문제에 대해 채점되는 기능입니다.

- 4가지의 solution 중 어느 부분이 완료되었는지 파악할 수 있는 기능입니다.
- 각 solution에 대해 구현이 완료되면 `raiseNotDefine()` 함수는 지우시면 됩니다.

```
Starting on 10-29 at 13:18:31

Question q1
=====

*** Method not implemented: depthFirstSearch at line 90 of search.py
*** FAIL: Terminated with a string exception.

### Question q1: 0/3 ###

Question q2
=====

*** Method not implemented: breadthFirstSearch at line 95 of search.py
*** FAIL: Terminated with a string exception.

### Question q2: 0/3 ###

Question q3
=====

*** Method not implemented: uniformCostSearch at line 100 of search.py
*** FAIL: Terminated with a string exception.

### Question q3: 0/3 ###

Question q4
=====

*** Method not implemented: aStarSearch at line 112 of search.py
*** FAIL: Terminated with a string exception.

### Question q4: 0/3 ###

Finished at 13:18:31

Provisional grades
=====
Question q1: 0/3
Question q2: 0/3
Question q3: 0/3
Question q4: 0/3
-----
Total: 0/12
```

Autograding

- 각 4가지 solution에 대해 테스트 결과, solution별 점수, 그리고 마지막에 최종 요약이 표시가 됩니다.
- 처음 실행하게 되면 모든 solution이 실패라고 출력이 됩니다(정상).
 - 각 solution을 해결하다 보면 해결한 solution은 통과하고 풀지 못한 solution은 실패가 나옵니다.
 - 모든 solution를 통과해야만 만점을 받습니다.

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- Q4: A* Search
- Submission

Setup

- 이 과제에서는 GPU가 필요하지 않기 때문에 colab이 아닌 개인 컴퓨터와 노트북을 통해 진행 하셔야 됩니다(환경 세팅은 이후 local setup에서 참조).
 - 코드를 실행하기 전 Anaconda 환경 세팅 이후 AI-24 환경을 실행해주세요.
 - 명령어: `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)

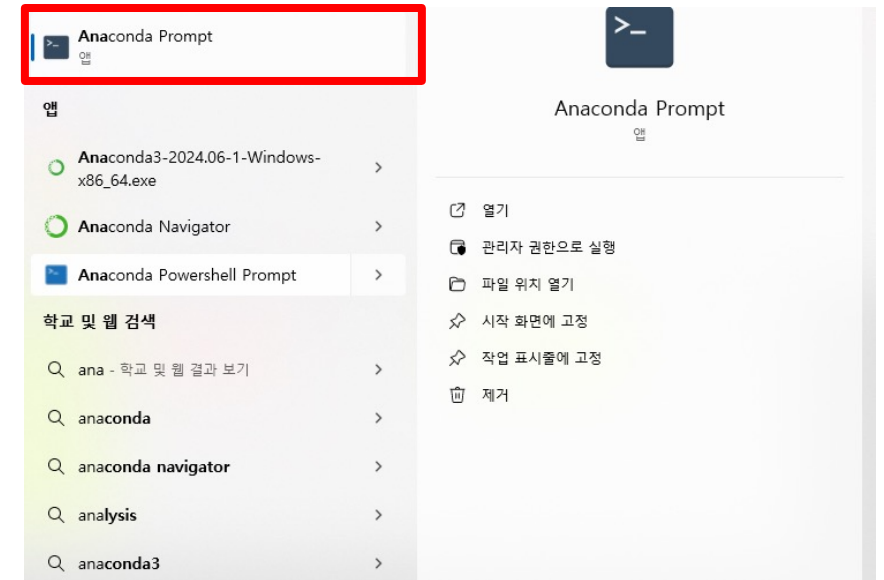
- Step 2 Join to Anaconda prompt and activate AI-24

- 1. conda env create -f environment.yml('cd'명령어를 사용해서 environment.yml파일이 있는 경로 접속 후 다음 명령어를 실행하세요.)

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

- 2. conda activate AI-24

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



Local setup

- Step 3 활성화 확인

- 모두 완료되면 코드를 실행할 수 있습니다!

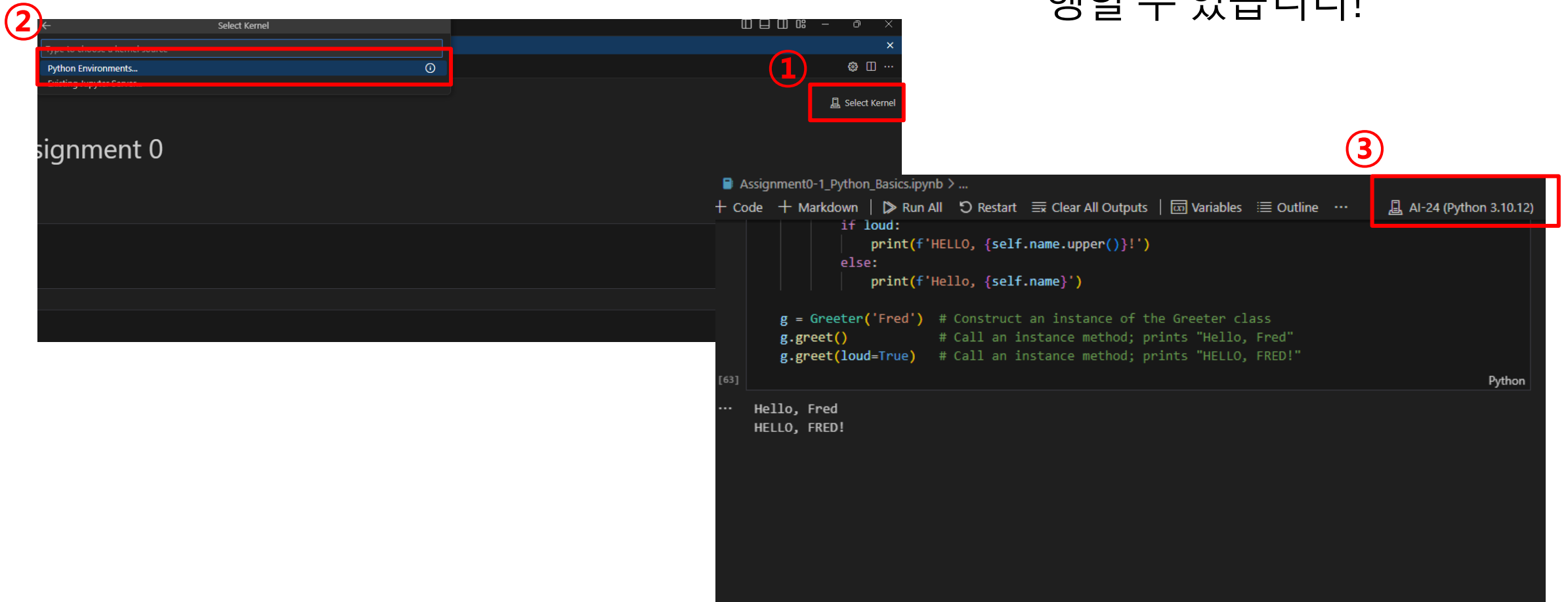


Table of Contents

- Introduction
- Setup
- **Welcome to Pacman**
- Q1: Depth First Search
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- Q4: A* Search
- Submission

Welcome to Pacman

- 코드(AS3_search.zip)를 다운로드하고 압축을 푼 후 해당 디렉토리로 이동하면, 다음 명령어를 커맨드 라인에 입력하여 `pacman` 게임을 실행할 수 있습니다:

```
python pacman.py
```

- Pacman은 푸른색의 구불구불한 통로와 둥근 먹이들이 있는 세계에서 살아가고 있으며, 이 미로 같은 세계를 효율적으로 탐색하는 것이 **Pacman이 자신의 영역을 마스터하기 위한 첫 번째 단계입니다.**

Welcome to Pacman

- `searchAgents.py` 가장 간단한 에이전트는 `GoWestAgent`로, 항상 서쪽으로 이동하는 간단한 반사형 에이전트입니다. 이 에이전트는 가끔씩 우승할 수 있습니다.

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

- 그러나, 방향 전환이 필요한 상황(코너가 있을 때)에서는 문제가 생깁니다.:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

- `pacman`이 감히게 되면, 터미널에서 `CTRL-C`를 입력하여 게임을 종료할 수 있습니다.

Welcome to Pacman

- 여러분의 에이전트는 `tinyMaze`뿐만 아니라 원하는 모든 미로를 해결할 수 있게 됩니다.
- 참고로 `pacman.py` 는 여러 옵션을 지원하며, 각 옵션은 긴 방식과 짧은 방식으로 표현될 수 있습니다. 모든 옵션과 기본값 목록을 확인하려면 이 명령어를 사용하세요:

```
python pacman.py -h
```
- 또한, 이 프로젝트에 나오는 모든 명령어들은 `commands.txt`파일에 있으므로 복사해서 붙여넣기가 쉽습니다. UNIX 또는 Mac OS X에서는 다음 명령어를 사용해서 이 모든 명령어들을 순서대로 실행 할 수 있습니다 `bash commands.txt`.

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- **Q1: Depth First Search**
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- Q4: A* Search
- Submission

Q1: Finding a Fixed Food Dot using Depth First Search

- `searchAgents.py`에는 `pacman`의 세계에서 경로를 계획하고 단계별로 실행하는 `searchAgent`가 완전히 구현되어 있습니다.
 - 하지만 계획을 세우기 위한 탐색 알고리즘은 아직 구현이 되어있지 않음 --구현해야할 과제
- 먼저 `SearchAgent`가 제대로 작동하는지 다음 명령어로 테스트하세요:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```
- 위 명령어는 `SearchAgent`에게 탐색 알고리즘으로 `tinyMazeSearch`를 사용하라고 지시하는 것으로, 이는 `search.py`에 구현이 되어 있습니다. `pacman`은 미로를 성공적으로 탐색해야 합니다.

Q1: Finding a Fixed Food Dot using Depth First Search

- 이제 pacman이 경로를 계획할 수 있도록 범용 탐색 함수를 작성할 차례입니다!
 - 작성할 탐색 알고리즘에 대한 의사 코드는 강의 노트에서 찾을 수 있습니다.
- 탐색 노드는 상태 뿐만 아니라 해당 상태로 도달하는 경로(계획)를 재구성하는 데 필요한 정보도 포함해야 한다는 점을 기억하세요.
- **중요한 점:** 모든 탐색 함수는 에이전트가 시작 지점에서 목표 지점으로 이동하는데 필요한 행동 리스트를 반환해야 합니다. 이 행동들은 모두 유효한 방향(벽을 통과하지 않음)이어야 하며 pacman이 미로를 성공적으로 탐색할 수 있어야 합니다.

Q1: Finding a Fixed Food Dot using Depth First Search

- `search.py`파일의 `depthFirstSearch`함수에서 깊이 우선 탐색 (DFS) 알고리즘을 구현하세요
 - 알고리즘을 완전하게 만들기 위해, 이미 방문한 상태를 다시 확장하지 않는 그래프 탐색 버전의 DFS를 작성하세요.
- **중요한 점:** 반드시 `util.py`에 제공된 `Stack`, `Queue`, `PriorityQueue` 데이터 구조를 사용하세요! 이 데이터 구조 구현은 autograding와의 호환성을 위해 특정한 속성을 가지고 있으므로 필수로 사용하셔야 합니다.

Q1: Finding a Fixed Food Dot using Depth First Search

- 해당 명령어를 통해 solution을 해결하면 됩니다.:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
```

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
```

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=dfs
```

- Pacman 보드는 탐색된 상태와 탐색 순서를 overlay하여 보여줍니다(밝은 빨간색일수록 더 먼저 탐색된 영역을 나타냅니다). 탐색 순서가 예상했던 대로 진행되나요? Pacman이 목표로 가는 과정에서 탐색한 모든 칸을 실제로 거쳤나요?

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- **Q2: Breadth First Search**
- Q3: Uniform Cost Search
- Q4: A* Search
- Submission

Q2: Breadth First Search

- `search.py`파일의 `breadthFirstSearch` 함수에서 너비 우선 탐색 (BFS) 알고리즘을 구현하세요. 이미 방문한 상태를 다시 확장하지 않는 그래프 탐색 버전의 BFS를 작성해야 합니다. 깊이 우선 탐색(DFS)와 동일한 방식으로 코드를 테스트하시면 됩니다.

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
```

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs
```

- BFS가 최소 비용 해결책을 찾았나요? 그렇지 않다면 다시 확인해보세요.

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- Q2: Breadth First Search
- **Q3: Uniform Cost Search**
- Q4: A* Search
- Submission

Q3: Varying the Cost Function

- BFS는 목표까지 가는 최소 행동 경로를 찾기만, 다른 측면에서 "최선"의 경로를 찾고 싶을 때도 있습니다. 예를 들어, `mediumDottedMaze` 와 `mediumScaryMaze`를 고려해보세요.
- 비용 함수를 변경하여 pacman이 다른 경로를 찾을 수 있도록 유도할 수 있습니다. 예를 들어, 유령이 많은 위험한 지역의 이동에는 더 높은 비용을 부과하고, 음식이 많은 지역의 이동에는 더 낮은 비용을 부과할 수 있습니다. 합리적인 pacman 에이전트는 이러한 비용 변화에 따라 자신의 행동을 조정하게 될 것입니다.

Q3: Varying the Cost Function

- `search.py` 파일의 `uniformCostSearch` 함수에서 균일 비용 탐색 (Uniform-Cost Search) 알고리즘을 구현하세요.
- 구현에 도움이 될만한 데이터 구조들이 있으니 `util.py` 파일을 확인해 보시기 바랍니다.

Q3: Varying the Cost Function

- 이제 다음 세 가지 레이아웃에서 성공적인 행동을 관찰할 수 있어야 합니다. 여기서 에이전트들은 모두 균일 비용 탐색(UCS) 에이전트이며, 사용되는 비용 함수만 다릅니다(에이전트와 비용 함수는 이미 작성되어 있습니다).:

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
```

```
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
```

```
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

- 참고:** StayEastSearchAgent 와 StayWestSearchAgent는 각각 매우 낮고 매우 높은 경로 비용을 가지게 됩니다. 이는 그들의 지수적 비용함수 때문입니다(searchAgents.py 파일에서 자세한 내용을 확인하세요).

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- **Q4: A* Search**
- Submission

Q4: A* search

- `search.py` 에 비어 있는 `aStarSearch` 함수에서 A* 탐색 알고리즘을 구현하세요
- A* 탐색은 휴리스틱 함수를 인자로 받습니다. 휴리스틱 함수는 두 개의 인자를 받는데, 첫 번째 인자는 탐색 문제의 상태이고, 두 번째 인자는 문제 자체입니다 (참고용 정보로 사용됩니다).
- `search.py` 파일에 있는 `nullHeuristic` 함수는 간단한 예시입니다. 이를 참고해서 A* 탐색을 구현해보세요

Q4: A* search

- A* 구현을 미로에서 고정된 위치로 가는 경로를 찾는 문제에 대해 테스트할 수 있습니다. 이미 구현된 맨해튼 거리 휴리스틱(`manhattanHeuristic` in `searchAgents.py`)을 사용해 테스트하세요.

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

- A*는 **균일 비용 탐색**보다 약간 더 빠르게 최적의 해결책을 찾는 것을 볼 수 있을 것입니다.

Table of Contents

- Introduction
- Setup
- Welcome to Pacman
- Q1: Depth First Search
- Q2: Breadth First Search
- Q3: Uniform Cost Search
- Q4: A* Search
- **Submission**

Autograding

- A각 solution을 해결하면서 모든 test를 통과해야 합니다.
 - 모든 test를 통과하면 해당 solution에 대해 만점을 받습니다.

```
***      solution length: 152
***      nodes expanded:      173
*** PASS: test_cases/q3/ucs_4_testSearch.test
***      pacman layout:      testSearch
***      solution length: 7
***      nodes expanded:      14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
***      solution:      ['1:A->B', '0:B->C', '0:C->G']
***      expanded_states:      ['A', 'B', 'C']

### Question q3: 3/3 ###

Question q4
=====

*** PASS: test_cases/q4/astar_0.test
***      solution:      ['Right', 'Down', 'Down']
***      expanded_states:      ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q4/astar_1_graph_heuristic.test
***      solution:      ['0', '0', '2']
***      expanded_states:      ['S', 'A', 'D', 'C']
*** PASS: test_cases/q4/astar_2_manhattan.test
***      pacman layout:      mediumMaze
***      solution length: 68
***      nodes expanded:      221
*** PASS: test_cases/q4/astar_3_goalAtDequeue.test
***      solution:      ['1:A->B', '0:B->C', '0:C->G']
***      expanded_states:      ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
***      solution:      ['1:A->C', '0:C->G']
***      expanded_states:      ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
***      solution:      ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***      expanded_states:      ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###

Finished at 1:38:15

Provisional grades
=====
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
=====
Total: 12/12
```

Files

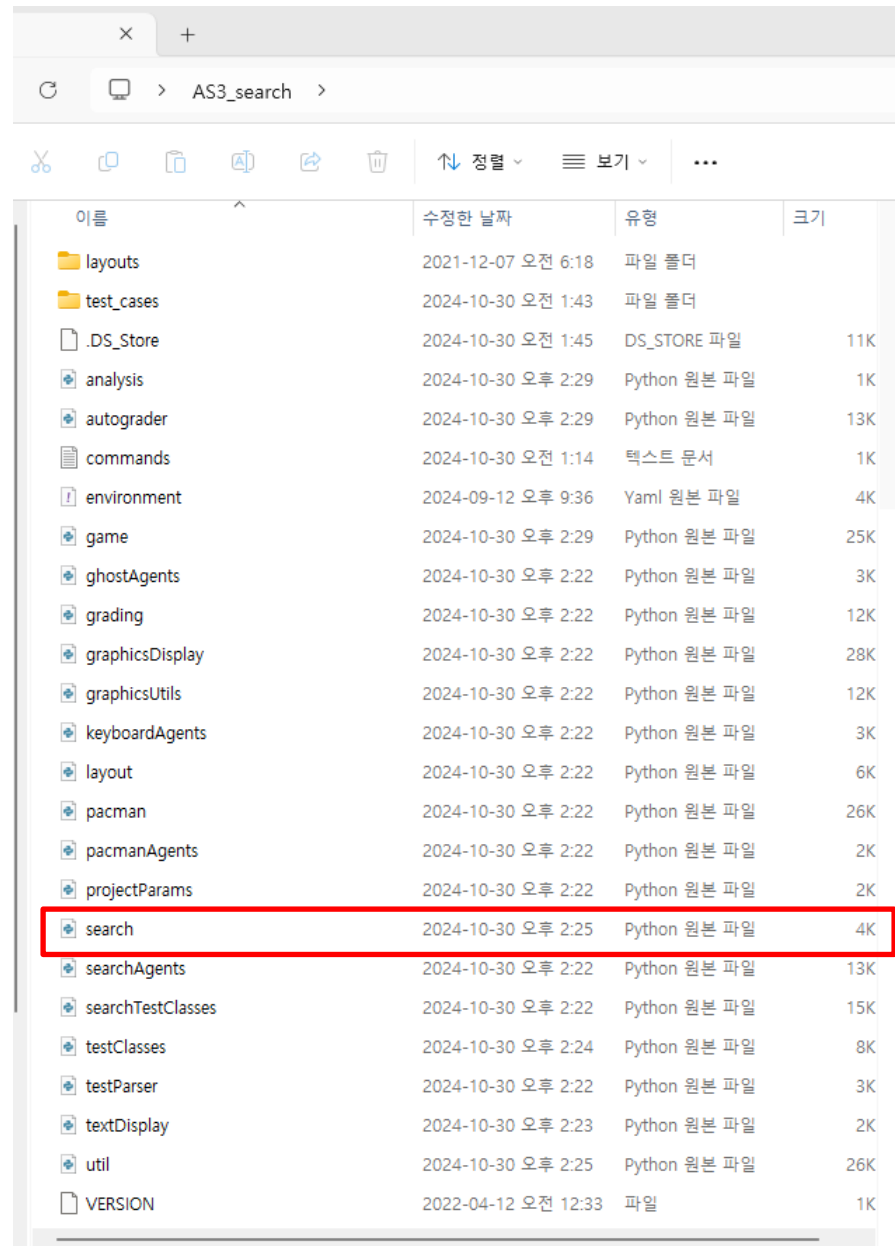
- 수정 및 제출해야 하는 파일:

- `search.py`: 모든 탐색 알고리즘은 이 파일에 구현하여 과제 완료 후 이 파일을 제출하면 됩니다.
- 해당 파일만 LMS과제 제출란에 올려 주시면 됩니다.

Files you'll edit:	
<code>search.py</code>	모든 탐색 알고리즘이 들어가야할 곳

Submitting your work

- Submitting your work
 - `search.py`
 - 이 파일을 수정해서 저장 후 LMS에 그대로 제출해주시면 됩니다.



A screenshot of a web-based file explorer interface. The address bar shows the path 'AS3_search'. The interface includes a toolbar with icons for file operations and a table listing files and folders. The table has four columns: '이름' (Name), '수정된 날짜' (Modified Date), '유형' (Type), and '크기' (Size). The file 'search' is highlighted with a red rectangular box.

이름	수정된 날짜	유형	크기
layouts	2021-12-07 오전 6:18	파일 폴더	
test_cases	2024-10-30 오전 1:43	파일 폴더	
.DS_Store	2024-10-30 오전 1:45	DS_STORE 파일	11K
analysis	2024-10-30 오후 2:29	Python 원본 파일	1K
autograder	2024-10-30 오후 2:29	Python 원본 파일	13K
commands	2024-10-30 오전 1:14	텍스트 문서	1K
environment	2024-09-12 오후 9:36	Yaml 원본 파일	4K
game	2024-10-30 오후 2:29	Python 원본 파일	25K
ghostAgents	2024-10-30 오후 2:22	Python 원본 파일	3K
grading	2024-10-30 오후 2:22	Python 원본 파일	12K
graphicsDisplay	2024-10-30 오후 2:22	Python 원본 파일	28K
graphicsUtils	2024-10-30 오후 2:22	Python 원본 파일	12K
keyboardAgents	2024-10-30 오후 2:22	Python 원본 파일	3K
layout	2024-10-30 오후 2:22	Python 원본 파일	6K
pacman	2024-10-30 오후 2:22	Python 원본 파일	26K
pacmanAgents	2024-10-30 오후 2:22	Python 원본 파일	2K
projectParams	2024-10-30 오후 2:22	Python 원본 파일	2K
search	2024-10-30 오후 2:25	Python 원본 파일	4K
searchAgents	2024-10-30 오후 2:22	Python 원본 파일	13K
searchTestClasses	2024-10-30 오후 2:22	Python 원본 파일	15K
testClasses	2024-10-30 오후 2:24	Python 원본 파일	8K
testParser	2024-10-30 오후 2:22	Python 원본 파일	3K
textDisplay	2024-10-30 오후 2:23	Python 원본 파일	2K
util	2024-10-30 오후 2:25	Python 원본 파일	26K
VERSION	2022-04-12 오전 12:33	파일	1K

