인공지능

Pacman Project

C184036 이소연

제출일 : 12월 16일 (토)

* Problem : Pacman은 특정 위치에 도달하고 음식을 효율적으로 먹기 위한 미로의 경로를 찾는 프로젝트 다음의 4가지 방법으로 프로그램 실행

- BFS

- DFS

- A*

- UniformCostSearch

<BFS>

- 알고리즘 코드

```python
### BFS ###
def breadthFirstSearch(problem):
    """Search the shallowest nodes in the search tree first."""

    # 큐를 사용하여 상태와 동작을 추적합니다.
    queue = util.Queue()

    # 시작 상태를 큐에 푸시합니다.
    start_state = problem.getStartState()
    queue.push((start_state, []))  # (상태, 동작 리스트)를 큐에 푸시합니다.

    # 방문한 상태를 저장하기 위한 집합을 생성합니다.
    visited = set()

    while not queue.isEmpty():
        state, actions = queue.pop()  # 상태와 현재까지의 동작 리스트를 꺼냅니다.

        # 현재 상태가 목표 상태인지 확인합니다.
        if problem.isGoalState(state):
            return actions  # 목표 상태에 도달한 경우 동작 리스트를 반환합니다.

        # 현재 상태를 방문한 것으로 표시합니다.
        visited.add(state)

        # 현재 상태에서 가능한 후속 상태들을 얻어옵니다.
        successors = problem.getSuccessors(state)

        for next_state, action, _ in successors:
            if next_state not in visited:
                next_actions = actions + [action]  # 현재 동작 리스트에 새로운 동작을 추가합니다.
                queue.push((next_state, next_actions))  # 다음 상태와 동작 리스트를 큐에 푸시합니다.

    # 목표 상태에 도달하지 못한 경우 None을 반환합니다.
    return None
```

- 2번 실행 결과

```
Path found with cost 117.
Pacman emerges victorious! Score: 1753
Average Score: 1753.0
Scores:        1753.0
Win Rate:      1/1 (1.00)
Record:        Win
```

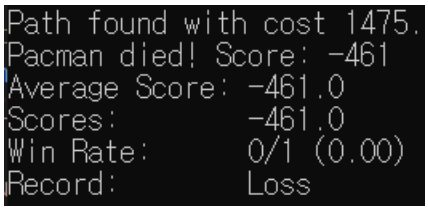
```
Path found with cost 117.
Pacman emerges victorious! Score: 1553
Average Score: 1553.0
Scores:        1553.0
Win Rate:      1/1 (1.00)
Record:        Win
```

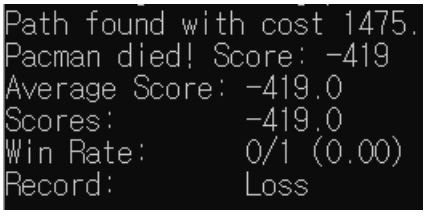
<DFS>

- 알고리즘 코드

```python
### DFS ###
def depthFirstSearch(problem):
    # 스택을 사용하여 상태와 동작을 추적합니다.
    stack = util.Stack()

    # 시작 상태를 스택에 푸시합니다.
    start_state = problem.getStartState()
    stack.push((start_state, []))  # (상태, 동작 리스트)를 스택에 푸시합니다.

    # 방문한 상태를 저장하기 위한 집합을 생성합니다.
    visited = set()

    while not stack.isEmpty():
        state, actions = stack.pop()  # 상태와 현재까지의 동작 리스트를 꺼냅니다.

        # 현재 상태가 목표 상태인지 확인합니다.
        if problem.isGoalState(state):
            return actions  # 목표 상태에 도달한 경우 동작 리스트를 반환합니다.

        # 현재 상태를 방문한 것으로 표시합니다.
        visited.add(state)

        # 현재 상태에서 가능한 후속 상태들을 얻어옵니다.
        successors = problem.getSuccessors(state)

        for next_state, action, _ in successors:
            if next_state not in visited:
                next_actions = actions + [action]  # 현재 동작 리스트에 새로운 동작을 추가합니다.
                stack.push((next_state, next_actions))  # 다음 상태와 동작 리스트를 스택에 푸시합니다.

    # 목표 상태에 도달하지 못한 경우 None을 반환합니다.
    return None
```

- 2번 실행 결과



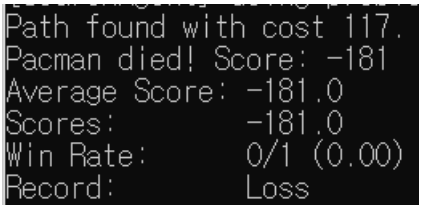


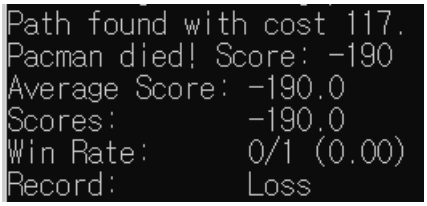
<A*>

- 알고리즘 코드

```python
### A* ###
def aStarSearch(problem, heuristic=nullHeuristic):
    # 시작 노드를 생성하고 초기 설정을 수행합니다.
    start_node = (problem.getStartState(), [], 0)  # (상태, 동작 리스트, 비용) 형태의 노드
    open_set = util.PriorityQueue()  # 우선순위 큐를 사용하여 열린 집합을 관리
    open_set.push(start_node, 0)  # 시작 노드를 우선순위 큐에 추가 (우선순위는 비용)
    closed_set = set()  # 닫힌 집합을 관리
    while not open_set.isEmpty():
        current_state, actions, current_cost = open_set.pop()  # 가장 우선순위가 높은 노드를 가져옴
        if problem.isGoalState(current_state):
            return actions  # 목표 상태에 도달한 경우 동작 리스트 반환
        if current_state not in closed_set:
            closed_set.add(current_state)  # 현재 상태를 닫힌 집합에 추가
            for next_state, action, step_cost in problem.getSuccessors(current_state):
                next_actions = actions + [action]  # 현재 동작 리스트에 새로운 동작 추가
                next_cost = current_cost + step_cost  # 경로 비용 업데이트
                priority = next_cost + heuristic(next_state, problem)  # 우선순위 계산
                open_set.push((next_state, next_actions, next_cost), priority)  # 우선순위 큐에 추가
    return None  # 목표 상태에 도달하지 못한 경우 None 반환
```

- 2번 실행 결과



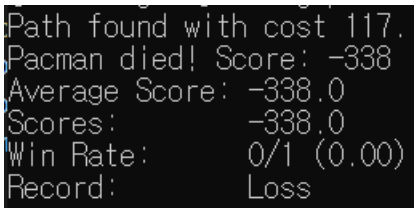


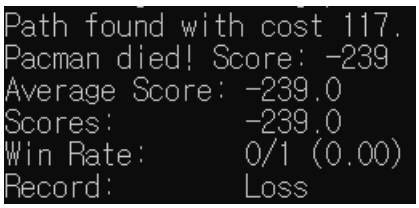
<UniformCostSearch>

- 알고리즘 코드

```python
### UniformCostSearch ###
def uniformCostSearch(problem):
    # 시작 노드를 생성하고 초기 설정을 수행합니다.
    start_node = (problem.getStartState(), [], 0)  # (상태, 동작 리스트, 비용) 형태의 노드
    open_set = util.PriorityQueue()  # 우선순위 큐를 사용하여 열린 집합을 관리
    open_set.push(start_node, 0)  # 시작 노드를 우선순위 큐에 추가 (우선순위는 비용)
    closed_set = set()  # 닫힌 집합을 관리
    while not open_set.isEmpty():
        current_state, actions, current_cost = open_set.pop()  # 가장 우선순위가 높은 노드를 가져옴
        if problem.isGoalState(current_state):
            return actions  # 목표 상태에 도달한 경우 동작 리스트 반환
        if current_state not in closed_set:
            closed_set.add(current_state)  # 현재 상태를 닫힌 집합에 추가
            for next_state, action, step_cost in problem.getSuccessors(current_state):
                next_actions = actions + [action]  # 현재 동작 리스트에 새로운 동작 추가
                next_cost = current_cost + step_cost  # 경로 비용 업데이트
                open_set.push((next_state, next_actions, next_cost), next_cost)  # 우선순위 큐에 추가
    return None  # 목표 상태에 도달하지 못한 경우 None 반환
```

- 2번 실행 결과





<결과>

제일 좋은 순으로 나열하자면

1. BFS

2. A*

3. UniformCostSearch

4. DFS