人工智能实验一

PB18000221 袁一玮

实验环境

Python 3.7.3、Kernel 4.19.181-1、Debian 10 (buster)

实验内容

BFS

类似 DFS，使用 `Queue` 来存储访问到的路径，`visited` 字典保存访问到的节点。

每次从队列中弹出节点，若是目标节点，则通过 visited 获取到到达目标节点的路径返回；如果该节点还未访问过，则将该节点加入 visited 中。循环上述过程直至队列为空。

```
def myBreadthFirstSearch(problem):
    visited = {}
    queue = util.Queue()
    queue.push((problem.getStartState(), None))
    while not queue.isEmpty():
        state, prev_state = queue.pop()
        if problem.isGoalState(state):
            solution = [state]
            while prev_state != None:
                solution.append(prev_state)
```

```
                prev_state = visited[prev_state]
            return solution[::-1]
        if state not in visited:
            visited[state] = prev_state
            for next_state, step_cost in problem.getChildren(state):
                queue.push((next_state, state))
    return []
```

A*

A* 算法使用优先队列来排序，按照 f(n)=h(n)+g(n) 排序。与 BFS 类似，每次从优先队列弹出第一个元素，在 `visited` 字典中保存访问的节点，`cost` 字典保存了到每个节点的距离。

初始时 `cost` 字典中只有初始节点，距离是 0；在新节点上使用 `cost[next_state] = cost[state] + step_cost` 记录距离。每次从优先队列弹出节点，若是目标节点则通过 `visited` 找到路径并返回。循环上述过程直至优先队列为空。

```
def myAStarSearch(problem, heuristic):
    visited = {}
    cost = {}
    pqueue = util.PriorityQueue()
    cost[problem.getStartState()] = 0.0 # start
    pqueue.push((problem.getStartState(), None), heuristic(problem.getStartState()))

    while not pqueue.isEmpty():
        state, prev_state = pqueue.pop()
        if problem.isGoalState(state):
            solution = [state]
            while prev_state != None:
                solution.append(prev_state)
                prev_state = visited[prev_state]
            return solution[::-1]
```

```
        if state not in visited:
            visited[state] = prev_state
            for next_state, step_cost in problem.getChildren(state):
                cost[next_state] = cost[state] + step_cost
                pqueue.push((next_state, state), heuristic(
                    next_state)+cost[next_state])
    return []
```

Minimax

```
def minimax(self, state, depth):
    if state.isTerminated():
        return None, state.evaluateScore()

    best_state = None
    if state.isMe():
        best_score = -float('inf')
    else:
        best_score = float('inf')
    if state.isMe():
        # print('Me depth:', depth)
        if depth == 0:
            return state, state.evaluateScore()

    for child in state.getChildren():
        if state.isMe():
            child_state, child_score = self.minimax(child, depth-1)
            if child_score > best_score:
                best_score = child_score
                best_state = child
        else:
            child_state, child_score = self.minimax(child, depth)
            if child_score < best_score:
```

```
                best_score = child_score
                best_state = child
    return best_state, best_score
```

Alpha-beta 剪枝

alpha-beta 剪枝与 Minimax 类似。在遍历决策树时，如果发现我方 Agent 找到的子节点的最大值大于 beta 时，遍历子节点无意义；同理，如果发现对方 Agent 找到的子节点最小值小于 alpha，也可以进行剪枝。

书上算法有坑，判断当前最大（小）值和 beta(alpha) 的关系时不要用 >=(<=) 而是要用 >(<)，否则可能会错误剪枝。

```
def alphaBeta(self, state, depth, alpha, beta):
    if state.isTerminated():
        return None, state.evaluateScore()

    best_state = None
    if state.isMe():
        best_score = -float('inf')
    else:
        best_score = float('inf')
    if state.isMe():
        # print('Me depth:', depth)
        if depth == 0:
            return state, state.evaluateScore()

    for child in state.getChildren():
        if state.isMe():
            child_state, child_score = self.alphaBeta(
                child, depth, alpha, beta)
            if child_score > best_score:
                best_score = child_score
                best_state = child
```

```
            if best_score > beta:
                return best_state, best_score
            alpha = max(alpha, best_score)
        else:
            if child.isMe():
                child_state, child_score = self.alphaBeta(
                    child, depth-1, alpha, beta)
            else:
                child_state, child_score = self.alphaBeta(
                    child, depth, alpha, beta)
            if child_score < best_score:
                best_score = child_score
                best_state = child
            if best_score < alpha:
                return best_state, best_score
            beta = min(beta, best_score)
    return best_state, best_score
```

实验结果

每次测试时运行 `bash test.sh`，test.sh 中测试了三个 search 和两个 agent 的策略

```
python3 search/autograder.py -q q1
python3 search/pacman.py -l mediumMaze -p SearchAgent --frameTime 0
python3 search/autograder.py -q q2
python3 search/pacman.py -l mediumMaze -p SearchAgent -a fn=bfs --frameTime 0
python3 search/autograder.py -q q3
python3 search/pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=manhattanHe
python3 multiagent/autograder.py -q q2
python3 multiagent/autograder.py -q q3
python3 multiagent/pacman.py -p AlphaBetaAgent -l mediumClassic --frameTime 0
```

所有 case 通过，尽管 agent 在最后一个测试时会失败

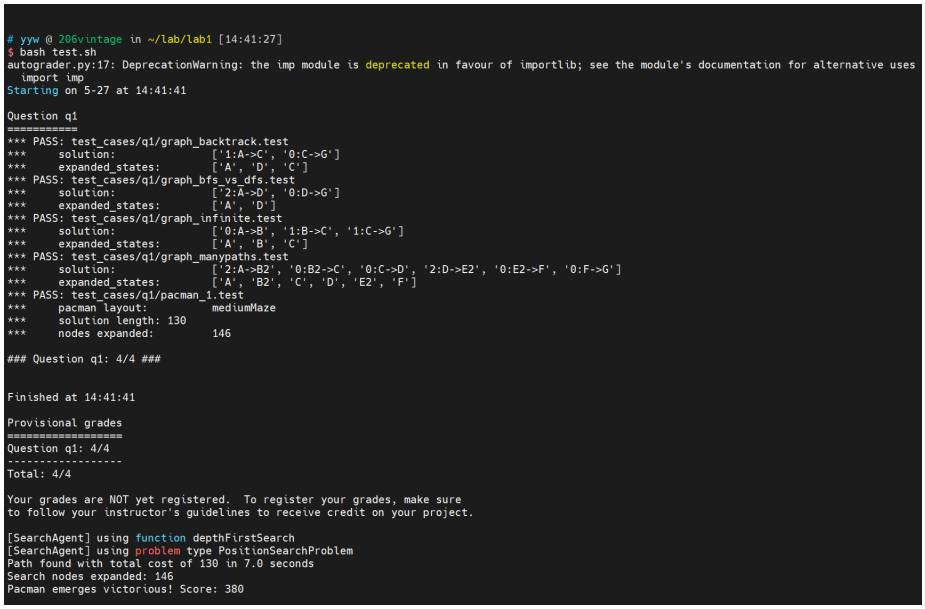
# yyw @ 206vintage in ~/lab/lab1 [14:41:27]
$ bash test.sh
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
import imp
Starting on 5-27 at 14:41:41
Question q1
==========
*** PASS: test_cases/q1/graph_backtrack.test
*** solution: ['1:A->C', '0:C->G']
*** expanded_states: ['A', 'D', 'C']
*** PASS: test_cases/q1/graph_bfs_vs_dfs.test
*** solution: ['2:A->D', '0:D->G']
*** expanded_states: ['A', 'D']
*** PASS: test_cases/q1/graph_infinite.test
*** solution: ['0:A->B', '1:B->C', '1:C->G']
*** expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q1/graph_manypaths.test
*** solution: ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
*** expanded_states: ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases/q1/pacman_1.test
*** pacman layout: mediumMaze
*** solution length: 130
*** nodes expanded: 146
Question q1: 4/4
Finished at 14:41:41
Provisional grades
==========
Question q1: 4/4
Total: 4/4
Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 7.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380

图 1: ai-test1

```
*** 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 24 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 13:08:24

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

图 2: ai-test2