人工智能基础 lab1

> PB18111707 吕瑞
>
> 2021/5/26

实验目标

本次实验有 2 个部分，分别是 Search 和 Multiagent。

Search 的目标是吃豆人仅仅是寻找食物，需要我们实现BFS 和 A* 两种静态查找算法。

Multiagent 的目标是吃完所有食物，同时避开鬼，这是在有对手的情况下，需要我们实现 minimax 和 alpha-beta 剪枝算法来给出下一步决策，使自己利益最大化。

实验环境

- 操作系统：windows10 (64 位)
- 环境：conda python = 3.6

实验内容

Part1 Search

BFS

广度优先搜索算法原理：先扩展根节点，接着扩展根节点的所有后继，然后再扩展他们的后继，以此类推。一般地，在下一层的任何结点扩展之前，搜索树上本层深度的所有节点都已经扩展过。

可以使用 FIFO 的队列数据结构实现边缘组织，以保证每次总是扩展深度最浅的结点。

实现代码：

```python
# 只需要将助教给的深度优先搜索示例代码中的数据结构由 Stack 改为 Queue 即可

def myBreadthFirstSearch(problem):
    # YOUR CODE HERE
    # util.raiseNotDefined()
    visited = {}
    frontier = util.Queue()

    frontier.push((problem.getStartState(), None))

    while not frontier.isEmpty():
        # 每次取出队头状态
        state, prev_state = frontier.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):
                # 孩子状态加到队尾
                frontier.push((next_state, state))

    return []

```

A* 搜索

A* 搜索算法原理：从当前边缘队列中，选择估算函数值 $f(x)$ 最小的结点扩展。每次扩展了结点 n 后，分别计算 n 的所有孩子结点的估算函数值，并将孩子结点压入边缘队列。

$f(x) = g(n) + h(n)$

$g(n)$：初始结点到这个结点 n 的路径损耗的总和。

$h(n)$：启发式函数（heuristic），结点 n 到目标结点的最低耗散路径的耗散估计值。

可以使用优先队列（PriorityQueue）的数据结构来实现边缘组织。

实现代码：

```
def myAStarSearch(problem, heuristic):
    # YOUR CODE HERE
    # util.raiseNotDefined()
    visited = {}
    g_score = {} # type:dict 存放已访问结点的实际路径耗散值

    frontier = util.PriorityQueue()

    frontier.update((problem.getStartState(),None),0)

    g_score[problem.getStartState()] = 0

    while not frontier.isEmpty():
        state,prev_state = frontier.pop()
        current_g_score = g_score[state]

        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
            # print(prev_state)
            for next_state, step_cost in problem.getChildren(state):
                g_n = current_g_score + step_cost # 计算起点到孩子状态的实际耗散路径
                h_n = heuristic(next_state) # 计算孩子状态到终点的预估耗散
                # add children g_score
                g_score[next_state] = g_n
                # f(n) = g(n) + h(n)
                frontier.update((next_state,state),g_n+h_n)

    return []

```

Part2 Multiagent

MiniMax

极小极大值算法原理：假设两个游戏者始终按照最优策略行棋，那么结点的极小极大值是对应状态的效用值。对于给定的选择，MAX 选择移动到有极大值的状态，MIN 选择移动到有极小值的状态。

算法实现：

- 从当前状态计算极小极大决策。
- 使用简单的递归计算每个后继的极小极大值。递归算法自上而下一直前进到树的叶子结点，之后随着递归回溯通过搜索树把极小极大值回传。
- 当限定深度 depth 减为 0 时，不再向下扩展结点。

> ”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。“
>
> 所以只有在最后一个鬼走完时，`depth--`，即 `children.isME() == True` 时，`depth--`。

实现代码：

```
class MyMinimaxAgent():

    def __init__(self, depth):
        self.depth = depth

    def minimax(self, state, depth):
        if state.isTerminated():
            return state, state.evaluateScore()
        if depth == 0:
            return state, state.evaluateScore()

        best_state, best_score = None, -float('inf') if state.isMe() else float('inf')

        for child in state.getChildren():
            # YOUR CODE HERE
            # util.raiseNotDefined()
            if child.isMe():
                # 递归终止条件
                res_state,res_score = self.minimax(child,depth-1)
                if best_score > res_score:
                    best_state = child
                    best_score = res_score
            elif state.isMe():
                # 自己走 -- max
                res_state,res_score = self.minimax(child,depth)
                if best_score < res_score:
                    best_state = child
                    best_score = res_score
            else:
                # 当前状态不是自己走，下一步也不是自己走 -- min
                res_state, res_score = self.minimax(child, depth)
                if best_score > res_score:
                    best_state = child
                    best_score = res_score

        return best_state, best_score

    def getNextState(self, state):
        best_state, _ = self.minimax(state, self.depth)
        return best_state

```

AlphaBeta 剪枝

$\alpha-\beta$ 剪枝原理：是基于 MiniMax 算法的改进。它通过剪枝思想来尽可能消除部分搜索树。

一般原则是，考虑在树中某处的结点 n，选手选择移动到该节点。如果选手在 n 的父节点或者更上层的任何选择点有更好的选择 m，那么在实际的搜索中就永远不会到达 n，此时可以裁剪它。

α：到目前为止路径上发现的 MAX 的最佳选择。

β：到目前为止路径上发现的 MIN 的最佳选择。

算法实现：

- $\alpha-\beta$ 搜索中不断更新 α,β 的值，他们作为参数回传
- 当某个结点的值分别比目前的 MAX 的 α 或者 MIN 的 β 值更差的时候裁剪此结点。

> 因此，只要 $\beta<\alpha$ ，就可以裁剪此节点。

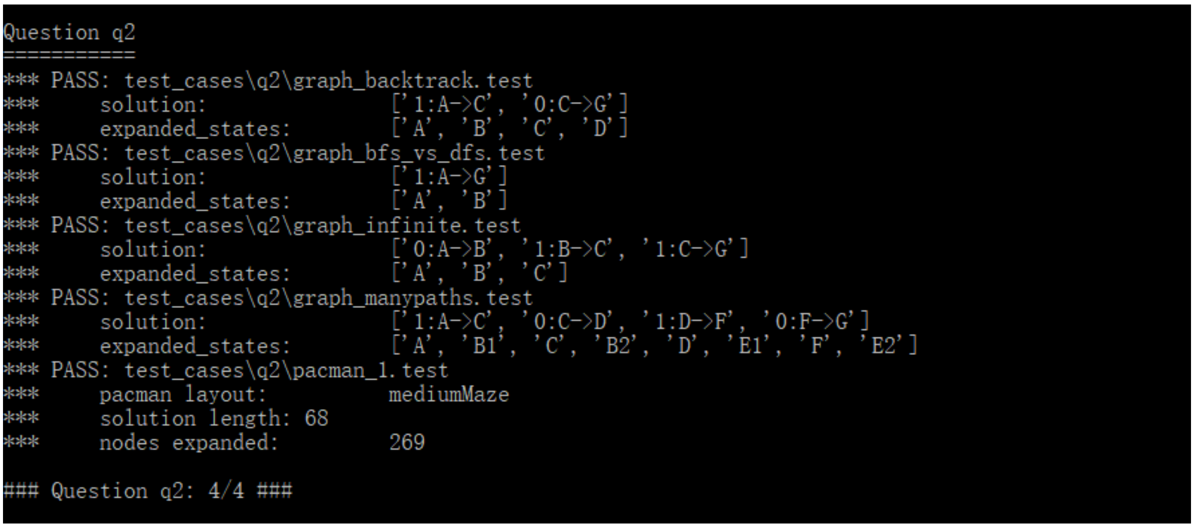
实现代码：

```
class MyAlphaBetaAgent():

    def __init__(self, depth):
        self.depth = depth

    def AlphaBeta(self,state,depth,a,b):

        if depth == 0 or state.isTerminated():
            return state, state.evaluateScore()

        best_state, best_score = None, -float('inf') if state.isMe() else float('inf')

        for child in state.getChildren():
            if child.isMe():
                # 递归终止条件 -- min
                temp_state, temp_score = self.AlphaBeta(child, depth-1,a,b)
                if best_score > temp_score:
                    best_state = child
                    best_score = temp_score
                # b = min(b, best_score)
                if best_score < a:
                    # 剪枝 当 a == b 时，允许继续探索一步
                    break
                b = min(b, best_score)

            elif state.isMe():
                # 自己走 -- max
                temp_state, temp_score = self.AlphaBeta(child, depth, a, b)
                if best_score < temp_score:
                    best_state = child
                    best_score = temp_score
                # a = max(a, best_score)
                if best_score > b:
                    # 剪枝 当 a == b 时，允许继续探索一步
                    break
                a = max(a, best_score)
            else:
                # 当前状态不是自己走，下一步也不是自己走 -- min
                temp_state, temp_score = self.AlphaBeta(child, depth, a, b)
                if best_score > temp_score:
                    best_state = child
                    best_score = temp_score
                # b = min(b, best_score)
                if best_score < a:
                    # 剪枝 当 a == b 时，允许继续探索一步
                    break
                b = min(b, best_score)

        return best_state, best_score

    def getNextState(self, state):
        # YOUR CODE HERE
        # util.raiseNotDefined()
        a = -float('inf')
        b = float('inf')
        best_state, _ = self.AlphaBeta(state, self.depth,a,b)
        return best_state

```

实验结果

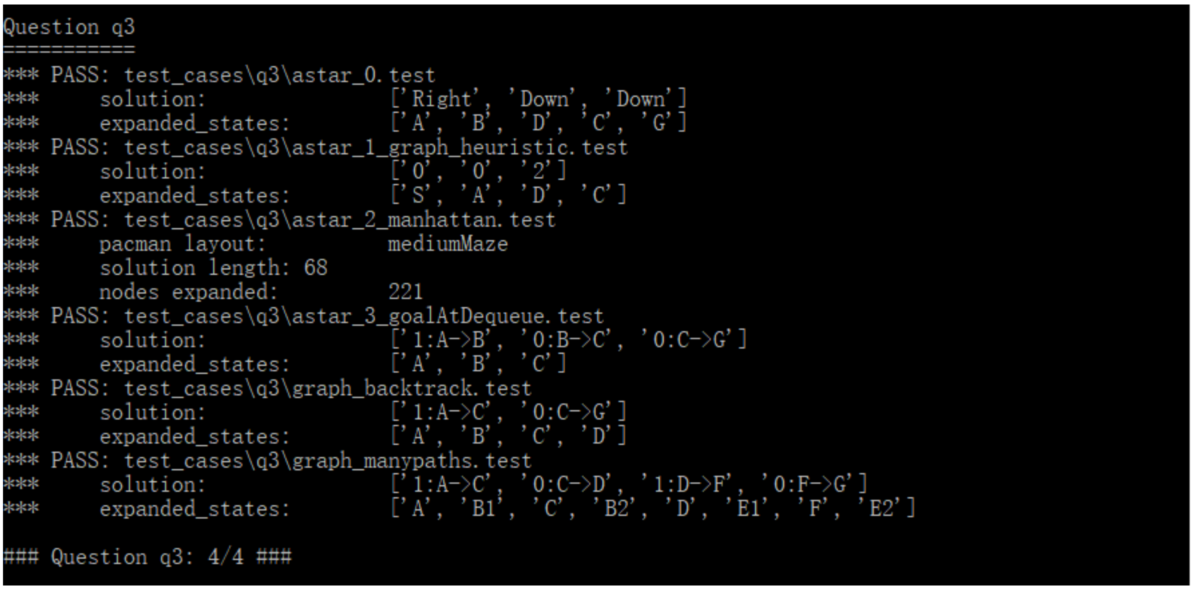
所有算法都能通过 test.sh

BFS:



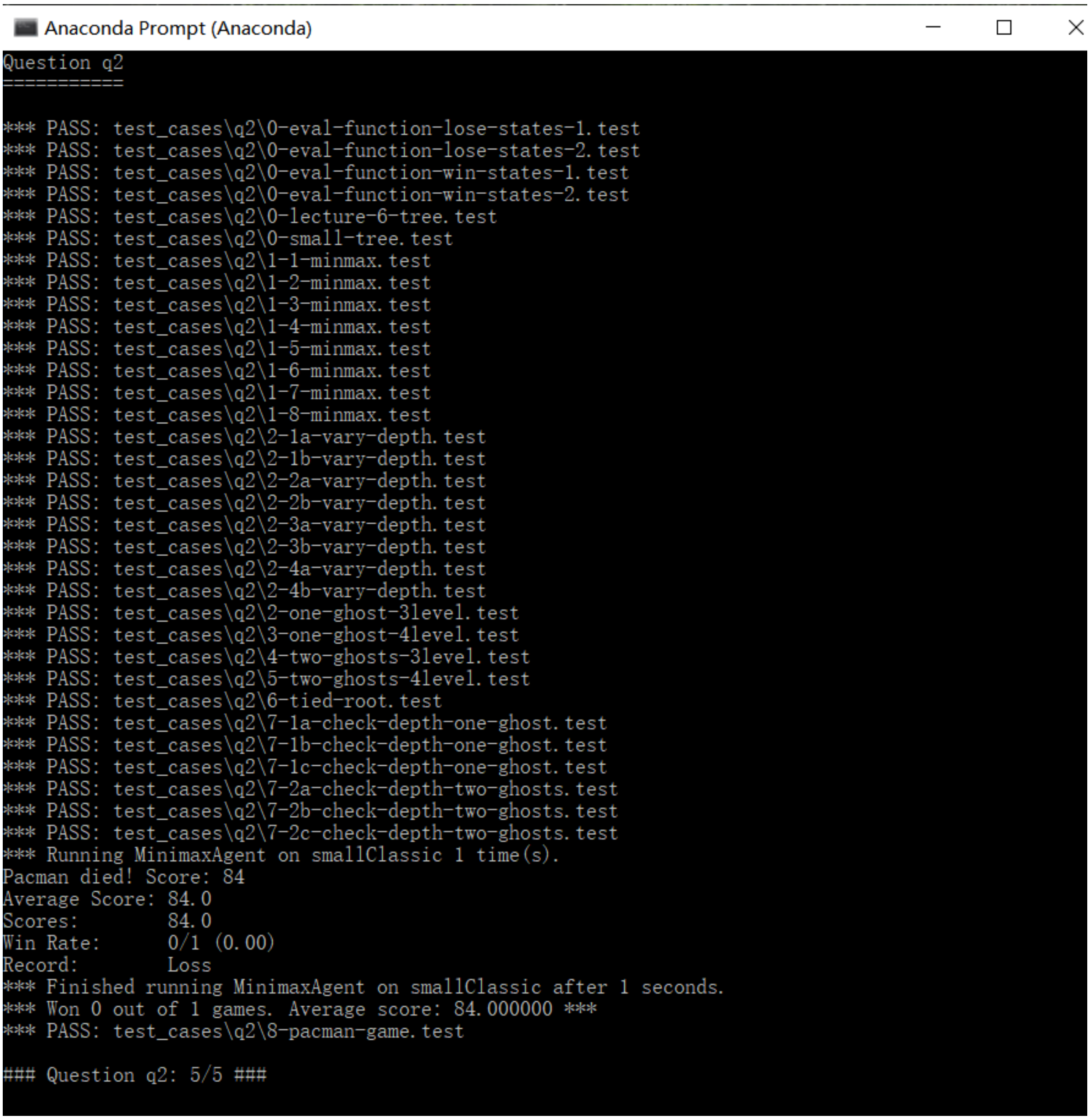
```
Question q2
===========
*** PASS: test_cases\q2\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q2\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q2\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q2\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']
*** PASS: test_cases\q2\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 4/4 ###
```

Astrar:



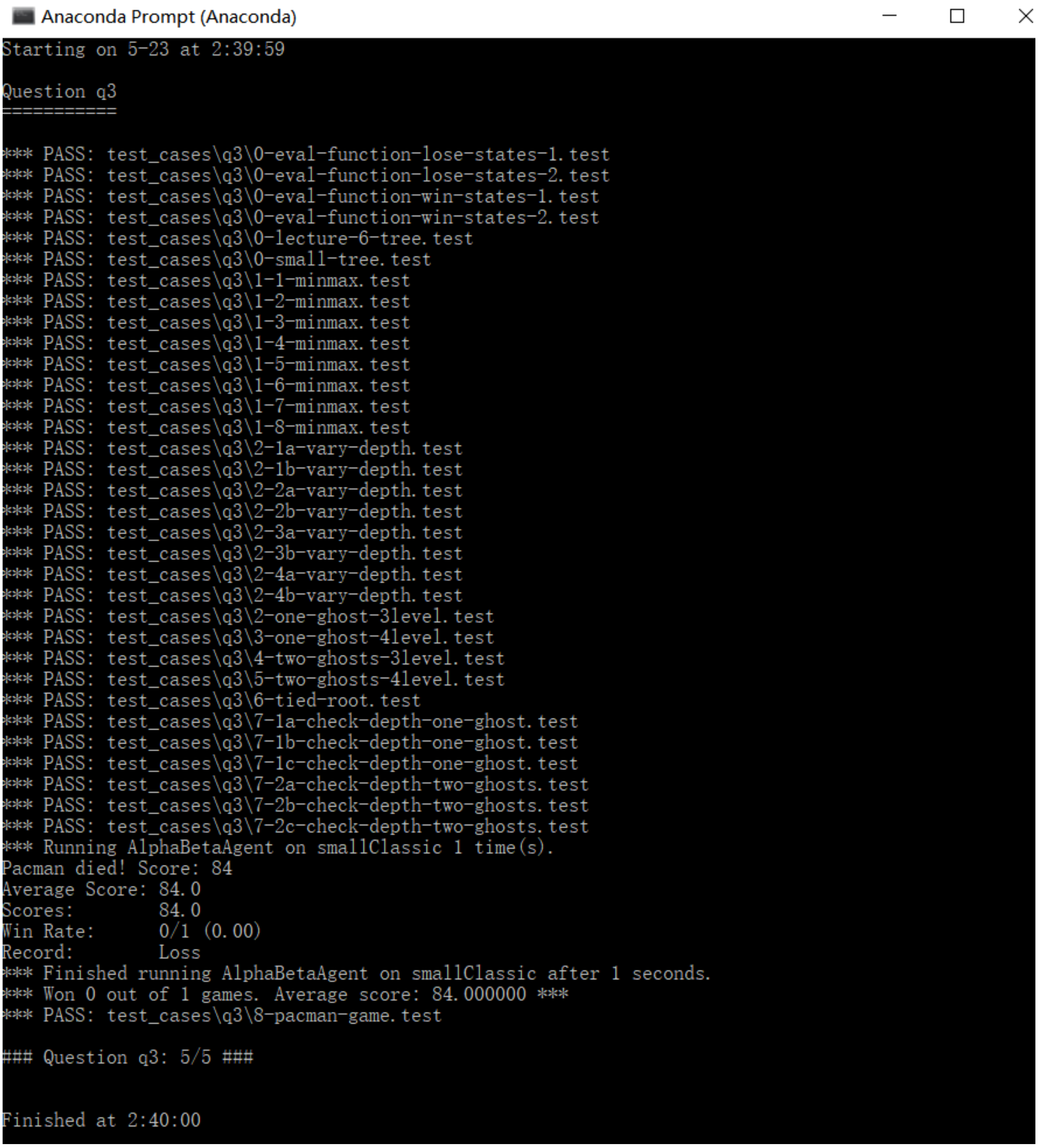
```
Question q3
===========
*** PASS: test_cases\q3\astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q3\astar_1_graph_heuristic.test
***     solution:               ['0', '0', '2']
***     expanded_states:        ['S', 'A', 'D', 'C']
*** PASS: test_cases\q3\astar_2_manhattan.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         221
*** PASS: test_cases\q3\astar_3_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q3\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q3\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 q3: 4/4 ###
```

MiniMax



```
Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent 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 MinimaxAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###
```

alpha-beta



```
Starting on 5-23 at 2:39:59

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
*** 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 1 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 2:40:00
```

python 语法糖记录

1. `print(array[::-1])` 表示将 array 数组中的元素倒序输出。
2. `PriorityQueue.pop()` 不会将比较标准 pop 出来。