人工智能基础LAB1-----实验报告

JL19110004 徐语林

一.实验介绍

本次实验是本学期人工智能基础课程的第一个实验，是对于搜索算法的一个应用。基于一个Pacman吃豆人的背景来感受BFS,DFS,A*,Minmax,Alpha-Beta算法，进一步加深对于上述搜索算法的理解。本次实验只需要补充上述算法即可完成所有任务。

二.实验内容及环境

实验分为了两个部分，分别是Search和Multiagent。第一个部分是针对静态查找算法，实现BFS,A*算法；第二个部分是博弈算法，实现Minmax以及Alpha-Beta剪枝

实验采用的是Ubuntu虚拟机，按照实验文档进行的环境配置。

三.实验过程

本次实验我还是按照了顺序先从静态算法再到博弈算法来进行实现。

静态搜索算法部分：

BFS算法：其实和在算法基础以及数据结构课程中的大体类似，就是使用一个队列，将最开始的结点先push入队列，然后按照队列先进先出的顺序来进行队列的扩展，以此来进行遍历。代码部分见附件，在此不做赘述。

A：其实在A*算法中踩了很多坑，原因是因为没有太弄清楚整个过程。A*算法相当于要使用一个优先队列的数据结构，优先级相当于是使用启发式函数算出来的估计值，加上前一段路径所耗费的真实值，每次pop()出来的是值最小的结点，再对此进行扩展即可。我的实现是用了一个cost的字典，来存放已经走过的路径所消耗的值，并不断对此字典进行更新。代码如下：

```
start=problem.getStartState()
    visited = {}
    cost = {} #存储已走过的cost
    frontier = util.PriorityQueue()
```

```
frontier.push((start, None),0)
mincost=0 #负责取前面走过的路径耗费值
ncost=0
cost[start]=0 #最开始为0
while not frontier.isEmpty():
    state, prev_state= frontier.pop()
    mincost=cost[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
        next_states=problem.getChildren(state)

        for next_state,step_cost in next_states:
            if next_state not in visited:
                cost[next_state]=step_cost+mincost #保存权值
                nextcost=cost[next_state]+heuristic(next_state)
                frontier.update((next_state, state),nextcost)
return []
```

`

博弈算法部分：

Minmax:对于Minmax算法，此部分增加了两个函数，一个函数用来处理如果搜索到了Max结点，一个函数用来处理如果搜索到了Min结点。对于Max结点对应的函数其实较为好处理一些，其实上就是先判断此时的状态是否已经到了一定的深度或者已经达到了终止状态，如果是这样的话就直接返回现在所获得的分数即可；如果不是便要进行进一步的搜索，由于只有一个吃豆人，因此吃豆人的孩子结点必定是对立的Min结点，因此就直接调用Min函数即可，此部分的关键代码如下：

```
for child in state.getChildren():
            value=self.min_value(child,depth)
            if value is not None and value>v:
                v=value
```

而对于另一部分Min结点的处理相对来说要复杂一些，前面一样也要判断此时所处状态是否是一个可以完结的状态，如果是的话，便直接返回游戏的分数即可；如果不是的话，要进行进一步的搜索，由于一个吃豆人的对立面可能不止一个Agent，因此要对现在后继结点的状态进行一个判断，如果还是Min结点的话，那么就调用处理Min结点的函数即可，如果不是Min结点，那么就调用处理Max的结点，此部分的关键代码如下：

```
#此处其实要注意的是Min的孩子结点不一定是max结点，因为不止有一个和吃豆人对立的Agent，因此要对孩子结点进行判断
        for child in state.getChildren():
            if child.isMe() == True:
                value=self.max_value(child,depth+1)
            else:
                value=self.min_value(child,depth)
            if value is not None and value<v:
                v=value
```

其余代码见附件，在此不做赘述。

Alpha-Beta算法：此部分要实现一个剪枝，其实大体的思路还是参考的上面的Minmax算法，分了两个具体的函数Max_value函数和Min_value函数。不同的是在原来Minmax算法的基础上，进行了改进，对于不需要搜索的状态进行了剪枝，Alpha为下界，Beta为上界，在max_value函数中如果当前的value值大于Beta,则可以直接返回，大于Alpha则需要更新Alpha的值；在min_value函数中如果当前的value<Alpha,则可以直接返回，小于Beta则需要更新Beta的值，关键代码如下：

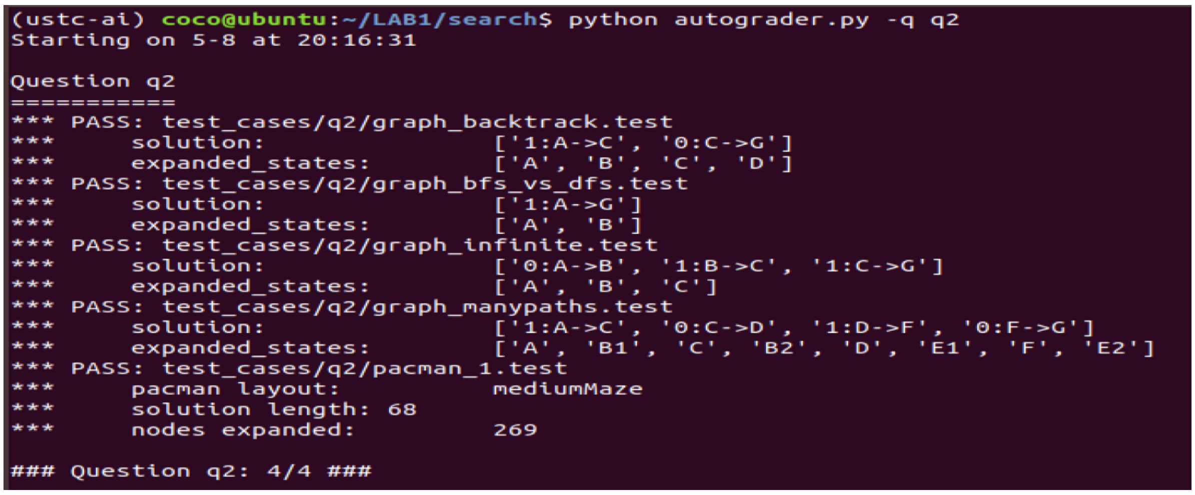
```
if value is not None and (maxvalue == None or value > maxvalue):
                    maxvalue=value
                    best_state=child
                #如果v>β，则直接返回v
                if value is not None and value>beta:
                    return value,child
                #更新α的值
                if value is not None and value>alpha:
                    alpha=value
```

```
if value is not None and (minvalue == None or value < minvalue):
                    minvalue=value
                    best_state=child
                #如果v<α，则直接返回v
                if value is not None and value<alpha:
                    return value,child
                #更新β的值
                if value is not None and value<beta:
                    beta=value
```

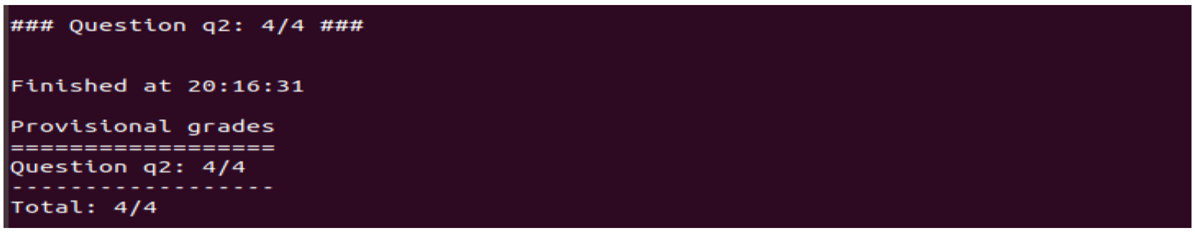
其余部分代码见附件。

四.实验结果

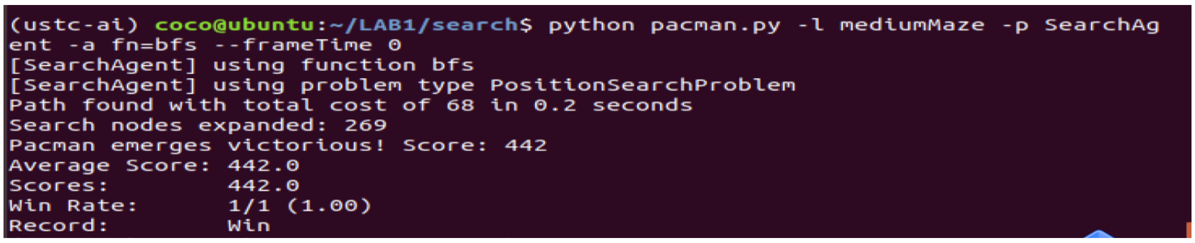
第一部分：BFS:



```
(ustc-ai) coco@ubuntu:~/LAB1/search$ python autograder.py -q q2
Starting on 5-8 at 20:16:31

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 ###
```

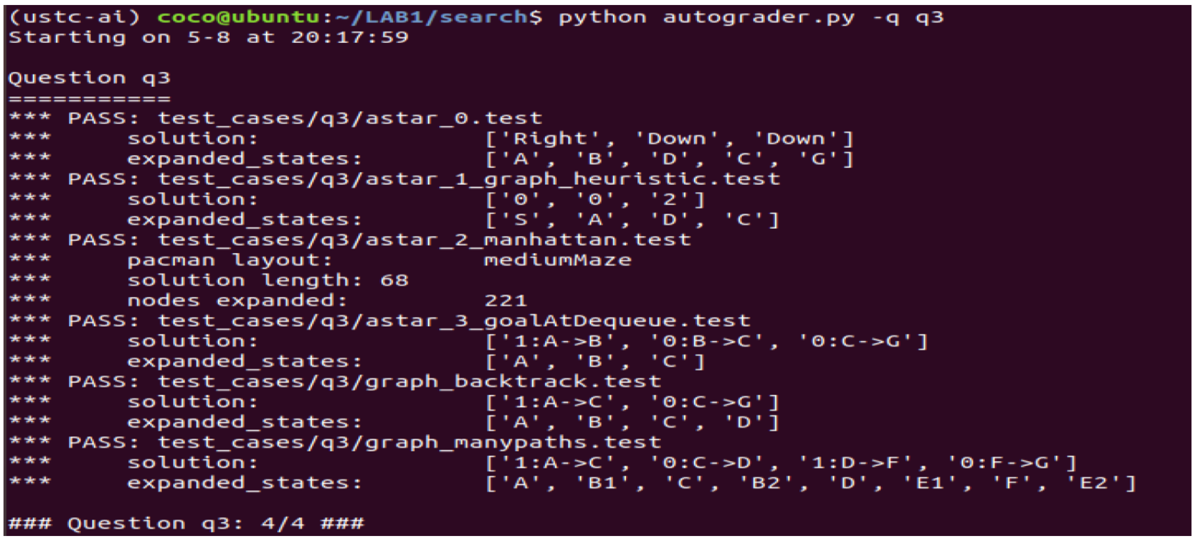


```
### Question q2: 4/4 ###


Finished at 20:16:31

Provisional grades
==================
Question q2: 4/4
------------------
Total: 4/4
```

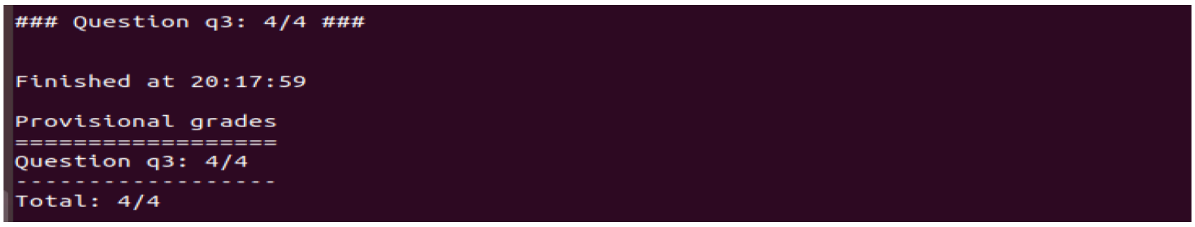


```
(ustc-ai) coco@ubuntu:~/LAB1/search$ python pacman.py -l mediumMaze -p SearchAg
ent -a fn=bfs --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.2 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

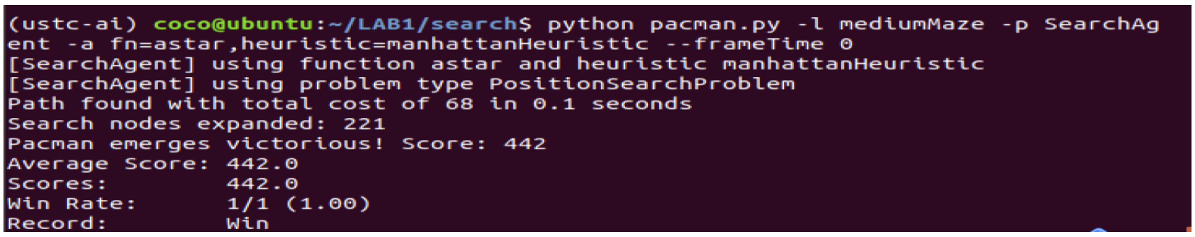
A*:



```
(ustc-ai) coco@ubuntu:~/LAB1/search$ python autograder.py -q q3
Starting on 5-8 at 20:17:59

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 ###
```

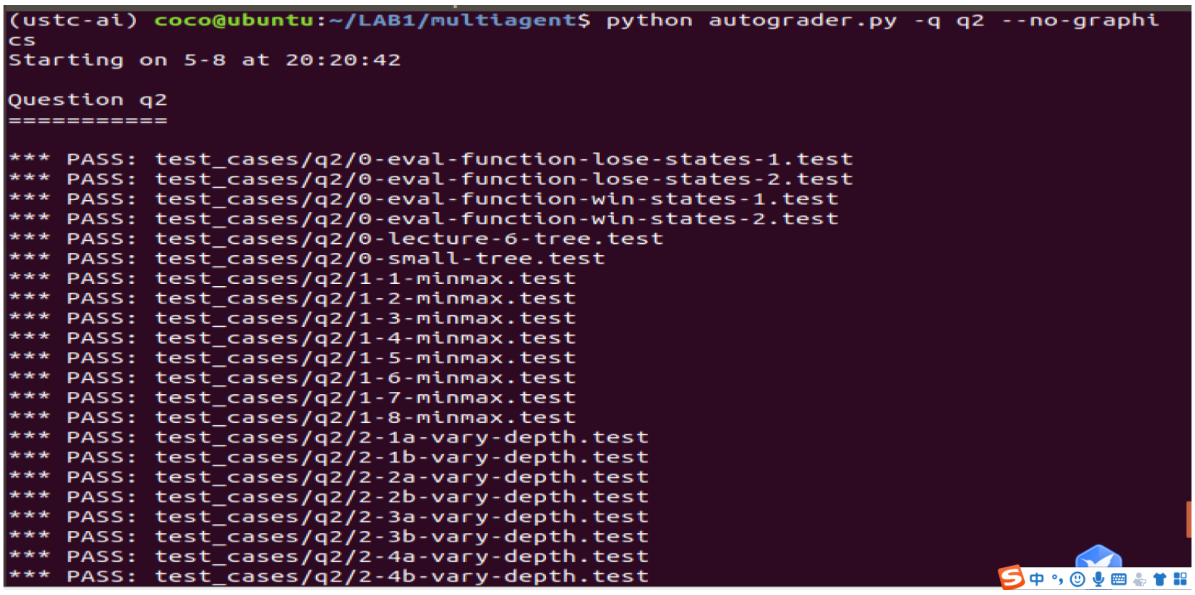


```
### Question q3: 4/4 ###


Finished at 20:17:59

Provisional grades
==================
Question q3: 4/4
------------------
Total: 4/4
```

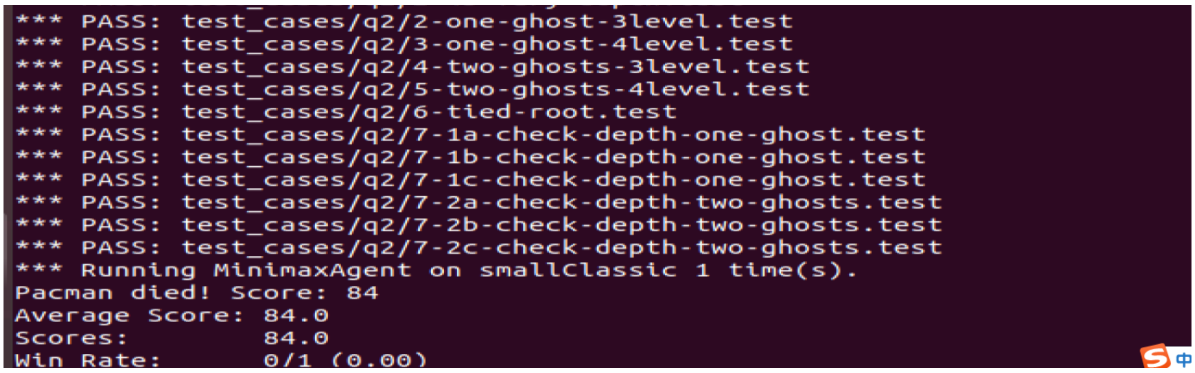


```
(ustc-ai) coco@ubuntu:~/LAB1/search$ python pacman.py -l mediumMaze -p SearchAg
ent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.1 seconds
Search nodes expanded: 221
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

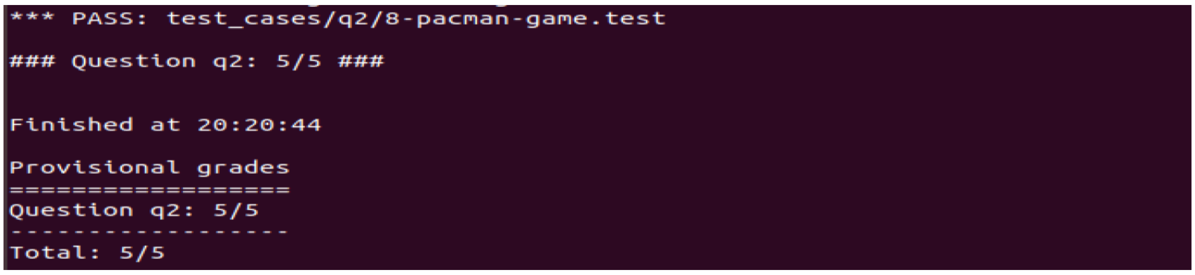
第二部分：Minmax:



```
(ustc-ai) coco@ubuntu:~/LAB1/multiagent$ python autograder.py -q q2 --no-graphi
cs
Starting on 5-8 at 20:20:42

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
```

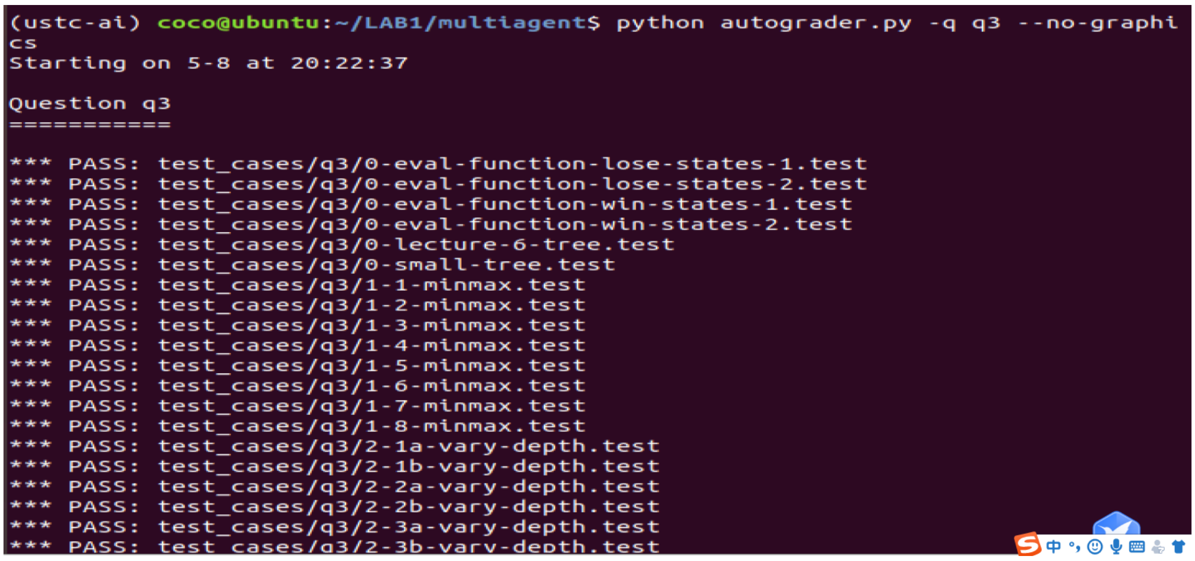


```
*** PASS: test_cases/q2/2-one-ghost-3level.test
*** PASS: test_cases/q2/3-one-ghost-4level.test
*** PASS: test_cases/q2/4-two-ghosts-3level.test
*** PASS: test_cases/q2/5-two-ghosts-4level.test
*** PASS: test_cases/q2/6-tied-root.test
*** PASS: test_cases/q2/7-1a-check-depth-one-ghost.test
*** PASS: 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)
```

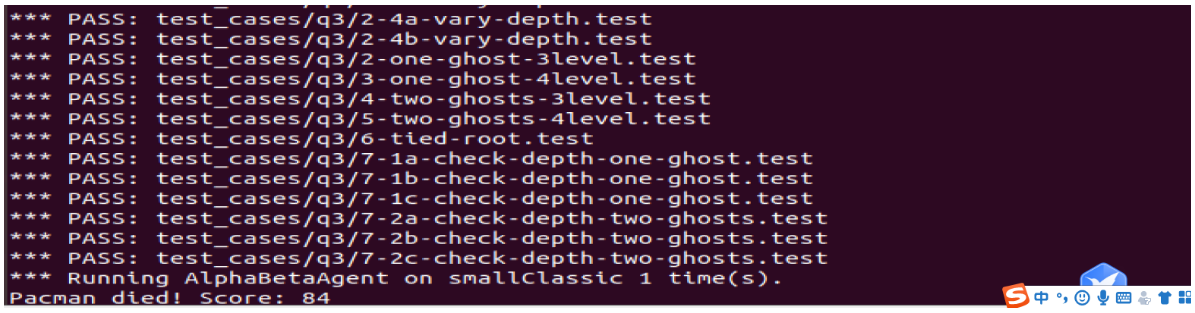


```
*** PASS: test_cases/q2/8-pacman-game.test

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


Finished at 20:20:44

Provisional grades
==================
Question q2: 5/5
------------------
Total: 5/5
```

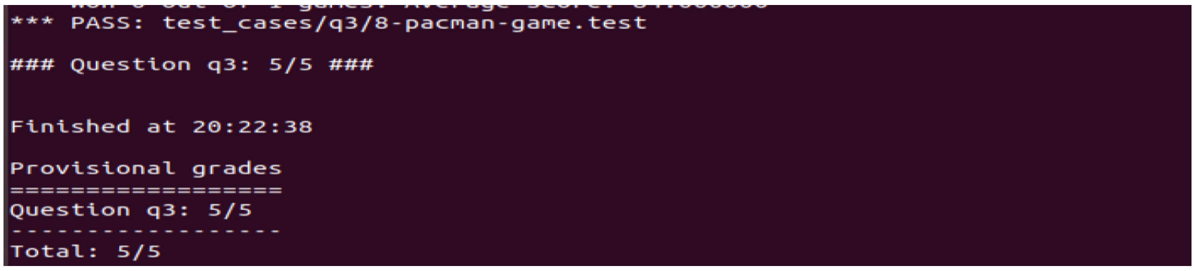
Alpha-Beta:



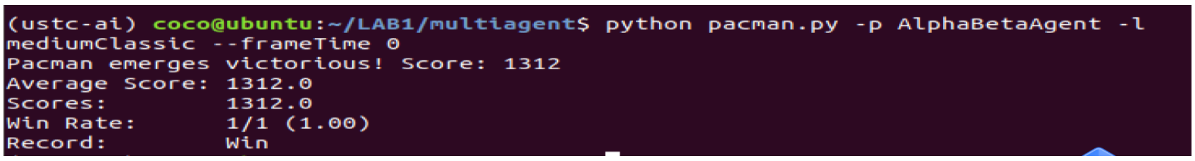
```
(ustc-ai) coco@ubuntu:~/LAB1/multiagent$ python autograder.py -q q3 --no-graphi
cs
Starting on 5-8 at 20:22:37

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
```



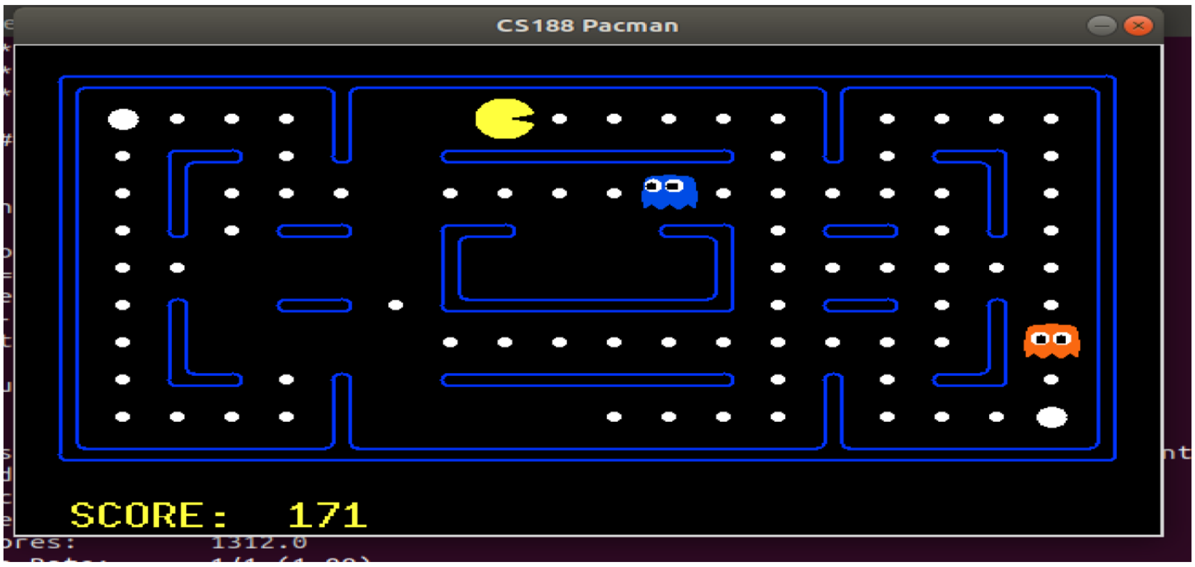
```
*** PASS: test_cases/q3/2-4a-vary-depth.test
*** PASS: test_cases/q3/2-4b-vary-depth.test
*** PASS: test_cases/q3/2-one-ghost-3level.test
*** PASS: test_cases/q3/3-one-ghost-4level.test
*** PASS: test_cases/q3/4-two-ghosts-3level.test
*** PASS: test_cases/q3/5-two-ghosts-4level.test
*** PASS: test_cases/q3/6-tied-root.test
*** 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
```



```
*** PASS: test_cases/q3/8-pacman-game.test

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


Finished at 20:22:38

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



```
(ustc-ai) coco@ubuntu:~/LAB1/multiagent$ python pacman.py -p AlphaBetaAgent -l
mediumClassic --frameTime 0
Pacman emerges victorious! Score: 1312
Average Score: 1312.0
Scores:        1312.0
Win Rate:      1/1 (1.00)
Record:        Win
```



五.实验总结

本次实验很好地让我理解了静态搜索算法以及博弈算法，本次实验在做的过程中主要的麻烦是在于对于python的生疏(以前学过但是又忘了一些)，导致在最开始写一行代码便会有一些莫名其妙的语法bug。但是助教已经把大多数实验中会用到的都已经在实验文档中给出，这也避免了在做的过程中反复的去查看有哪些函数可以用，这一点应该是大大减少了本次实验完成的时间。所以这次实验还重新让我熟悉了python的使用。