Lab1实验报告

实验内容

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；Multiagent的目标是吃完所有食物，同时避开鬼。Search部分需要实现BFS算法和A*算法。Multiagent部分需要实现minimax算法和alpha-beta剪枝。

实验环境

python3.6.13 环境， VSCode.

实验过程

BFS算法

- 算法设计

  广度优先搜索，使用队列。

```
while not frontier.isEmpty(): #队列不空
    state, prev_state = frontier.pop() #弹出
    if problem.isGoalState(state):   #如果搜索到，则停止
        done
    if state not in visited:         #该节点未搜索过
        visited[state] = prev_state
        for next_state, step_cost in problem.getChildren(state): #子节点入队
            frontier.push((next_state, state))
```

A*算法

- 算法设计

  伪代码加注释如下：

```
while len(open_list):
    #在open_list 里找最好的估计值 ， 记为 min_node
    min_state  = open_list[0]
    for state in open_list:
        if(f[state] < f[min_state]):
            min_state = state
    #如果结束.则输出
    if(problem.isGoalState(min_state)):
        done
    #遍历best_node 的子节点
    for child_state, step_cost in problem.getChildren(min_state):
        child_state_g = step_cost + g[min_state]
        if(child_state in close_list): #child_state 在 close_list 直接结束
```

```
                continue
            elif(child_state in open_list): #child_state 在 open_list , 比较是否需要更
新 。
                if(child_state_g < g[child_state]):
                    update
            else:           #child_state 都不在 ， 则加到open_list
                open_list.append(child_state)
        #刚处理过的节点 删除
        close_list.append(min_state)
        open_list.remove(min_state)
```

minimax算法

- 算法设计

  伪代码：

```
def minimax(self, state, depth):
        if state.isTerminated():      #终态 ， 计算分数，返回
            return None, state.evaluateScore()
        best_state, best_score = None, -float('inf') if state.isMe() else
float('inf')
        for child in state.getChildren():
            确定 child_score
            if state.isMe():
                    #找最大的
                best_score = max(best_score,child_score)
            else:
                    #找最小的
                best_score = min(best_score,child_score)
        return best_state, best_score
```

  确定 child_score 的方法：

  根据child节点的类型， 递归调用minimax 求child_score 。 如果是Pacman ， depth-1 。 如果是Ghost , depth不变 。 到达depth=1时 ， 直接计算Score 。

alpha-beta剪枝算法

- 算法设计

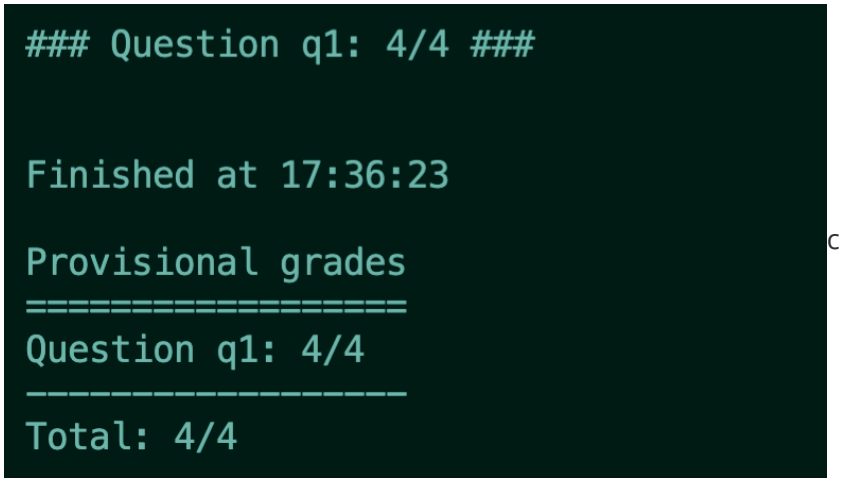
  伪代码

```
    def AlphaBeta(self,state,depth,a,b):
        if( state.isTerminated()): #终态 ， 计算分数，返回
            return None, state.evaluateScore()
        best_state, best_score = None, -float('inf') if state.isMe() else
float('inf')
```

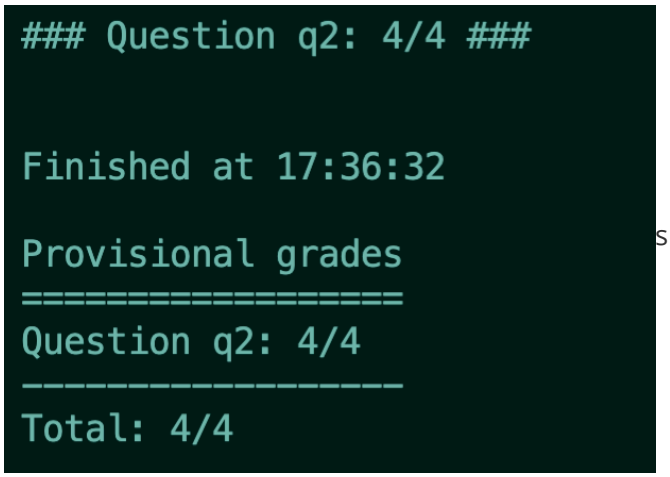
```
        for child in state.getChildren():
            # state 极大值点 。 找最大的子节点 ，  是作为 a 的最小值
                   确定 child_score
            if state.isMe():
                best_score = max(best_score,child_score)
                if best_score > b :
                    break       #b剪枝
                a = max(a,best_score)
            else:
                # state 极小值点 。 找最小的子节点 ，  是作为 b 的最大值
                best_score = min(best_score,child_score)
                if best_score  < a:
                    break       #a剪枝
                b = min(b,best_score)
        return best_state,best_score
```

实验结果

运行 ./test.sh



c



s

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


Finished at 17:36:39

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

s

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


Finished at 17:36:47

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

s

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


Finished at 17:36:48

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

s

实验总结

本次实验只需简单地实现两个搜索算法， 和在给定了`state.isTerminated()` ，`state.isTerminated()`，`state.evaluateScore()` 方法的情况下写两个agent 。 BFS算法只需， 类比DFS ，将栈改为队列即可。A*也是类似，不过是使用最小优先队列 。 两个agent算法最需注意depth的处理， 当是pacman agent时，depth-1 ， 当是Ghost agent时，depth不变。