AI-lab1

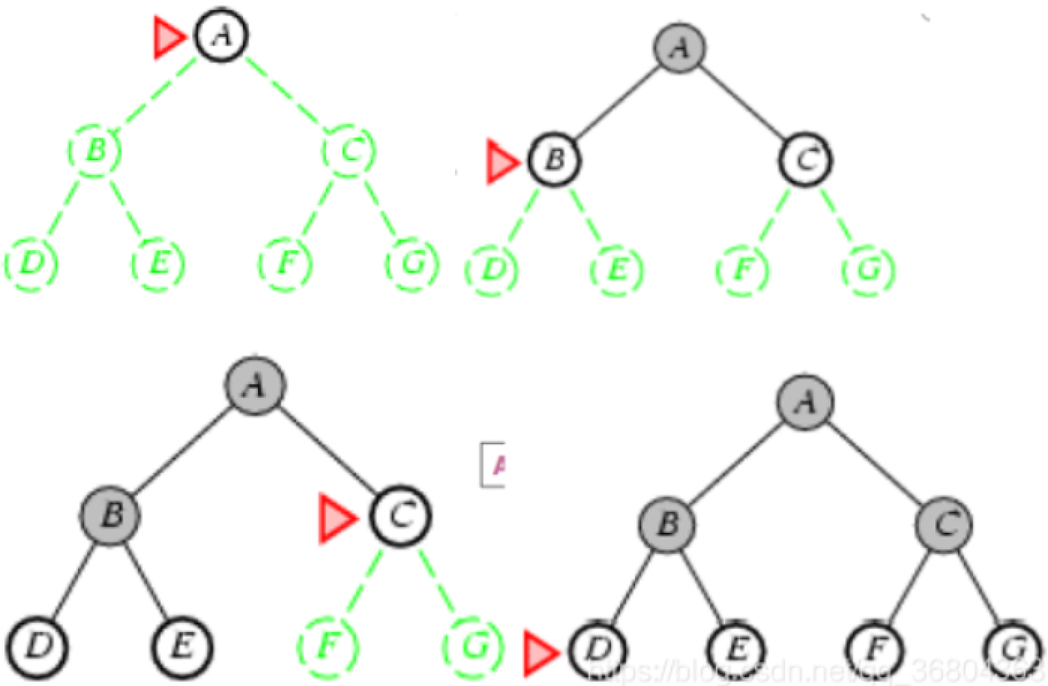
雷雨轩 PB18111791

Search

BFS

算法介绍

- 即宽度优先搜索，即从根节点开始扩展，接着扩展根节点的所有直接后继，然后在扩展这些直接后继的后继，依次类推。在下一层结点扩展前，搜索树上本层深度的结点都已扩展完毕
- 主要的实现思路是利用队列来实现，即FIFO，根节点最先进也最先扩展，然后依次出队的是直接后继，然后才是后继的后继。



- 书上伪代码

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

- 宽度优先搜索是完备的

- 若可以找到目标节点，那么一定是最浅的目标节点，最浅的目标节点不一定就是目标节点。如果路径是非递减函数，宽度搜索是最优的
- 时间复杂度：结点后继为b个，树深度为d，则最坏为 $b+b^2+...+b^d=O(b^d)$
- 空间复杂度：假定是把所有已扩展的结点都作保存，所以复杂度仍为$O(b^d)$，由边缘结点集决定

实验过程

- 实现思路比较简单，而且以前程序设计也写过类似的代码，所以只需要清楚实验所提供的接口，即可轻松实现

```
def myBreadthFirstSearch(problem):
    # YOUR CODE HERE
    visited={}
    Queue=util.Queue() #BFS需要用队列
    Queue.push((problem.getStartState(),None))

    while not Queue.isEmpty():
        state , prev_state = Queue.pop()
        if problem.isGoalState(state):#已到达目标结点
            solution = [state]
            while prev_state != None:
                solution.append(prev_state)
                prev_state = visited[prev_state]
            return solution[::-1]
        if state not in visited:#如果该结点没访问，则访问该结点，并把其邻居全部入队
            visited[state]=prev_state
            for next_state,step_cost in problem.getChildren(state):
                Queue.push((next_state,state))
    return []
```

- 代码与DFS非常相似，唯一不同的地方在于，BFS用的是队列，而DFS用的是栈，所以结点访问次序有区别

A*

算法介绍

- A*算法的评价函数为：f(n)=g(n)+h(n)，即同时计算了开始结点到当前结点已花费的代价，以及从当前结点到目标结点的估计代价
- 若想找到最小代价的解，扩展最小的f(n)是合理的，所以A*算法完备且最优
- 保证最优性的条件：可采纳性和一致性
    - 若h(n)可采纳，则A*的树搜索版本是最优的
    - 若h(n)一致，则A*的图搜索算法是最优的

实验过程

- 因为存在f(n)值的比较，所以考虑采用优先队列，队列中存放的元素为 `((state,prev_state,cost),f(n))`

  代码如下

```
def myAStarSearch(problem, heuristic):
    # YOUR CODE HERE
```

```python
    visited={}#维护每个状态的前驱结点
    pq=util.PriorityQueue()
    start_state=problem.getStartState()
    cur_cost=0#记录从初始状态出发,到当前结点的总代价
    pq.update((start_state,None,0),0+heuristic(start_state))
    while not pq.isEmpty():
        state,prev_state,cur_cost=pq.pop()
        if problem.isGoalState(state):#是目标结点，则返回其路径
            solution = [state]
            while prev_state is not 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):
                next_cost=cur_cost+step_cost

pq.update((next_state,state,next_cost),next_cost+heuristic(next_state))
    return []
```

- 思路大致为，对每一次出队的元素
    - 若为目标结点，则根据visited字典以及prev_state进行回溯，得到结果
    - 否则，若该状态尚未访问，那么标记为已访问，并且遍历其直接邻居，每个邻居n的f(n)值计算公式即`cur_cost+step_cost+heuristic(next_state)`

结果分析

- BFS测试结果：

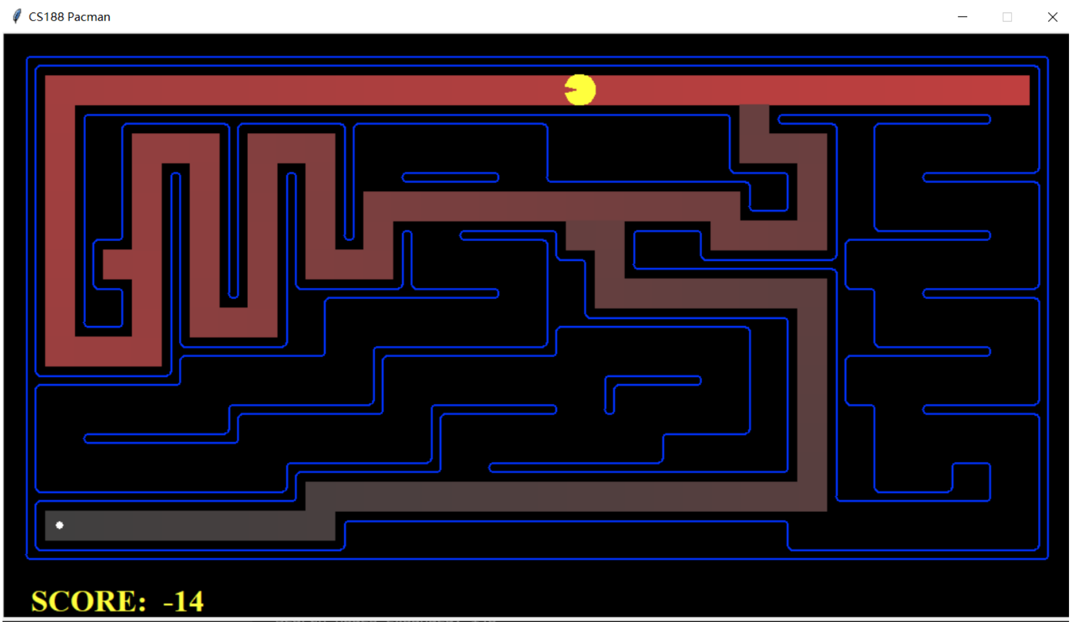
```
(ustc-ai) D:\科大\大三下\人工智能基础\lab\LAB1\search>python autograder.py -q q2
Starting on 5-28 at 21:59:56

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 ###


Finished at 21:59:56

Provisional grades
==================
Question q2: 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.
```

- A*测试结果：

```
(ustc-ai) D:\科大\大三下\人工智能基础\lab\LAB1\search>python autograder.py -q q3
Starting on 5-28 at 22:01:52

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 ###


Finished at 22:01:52

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

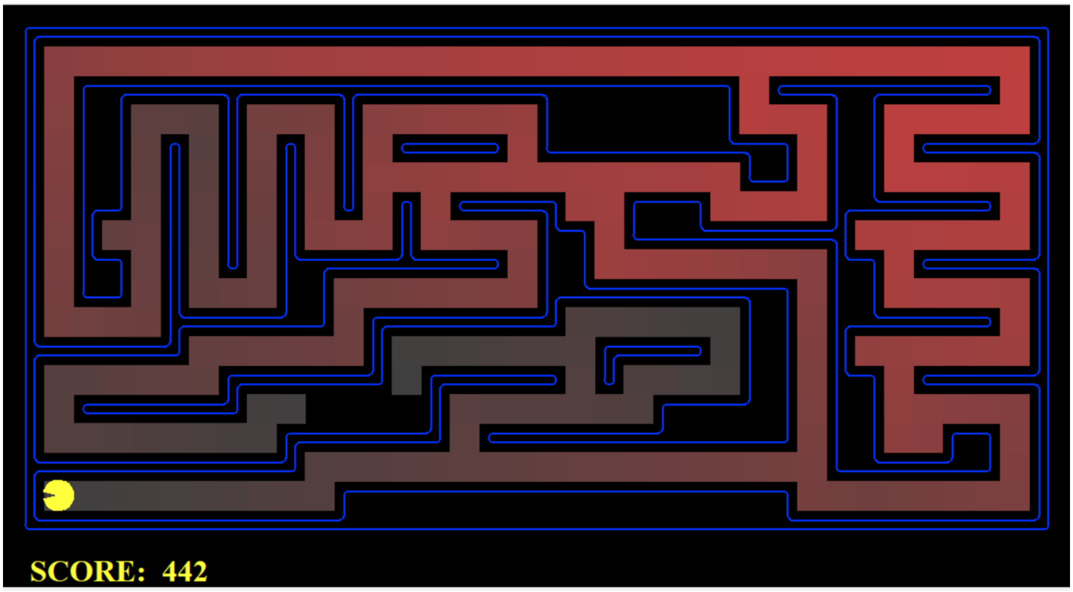
- DFS，BFS，A*结果比较
  - DFS

    Path found with total cost of 130 in 0.0 seconds
    Search nodes expanded: 146
    Pacman emerges victorious! Score: 380

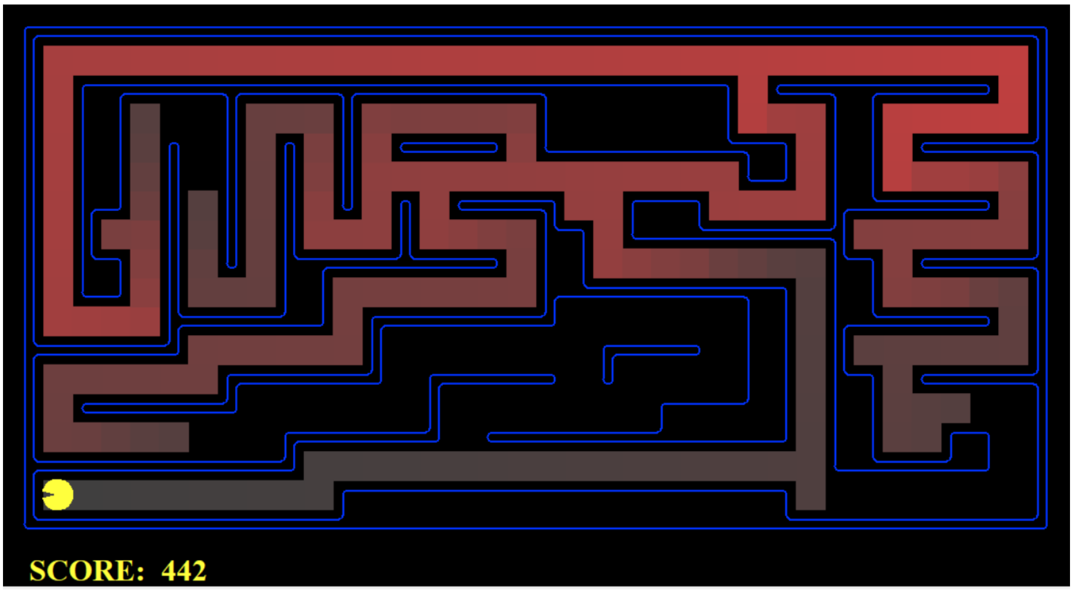
  - BFS

Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442



- A*

  Path found with total cost of 68 in 0.0 seconds
  Search nodes expanded: 221
  Pacman emerges victorious! Score: 442



对比结果发现，BFS扩展的结点最少，但是走了最长、最深的路径；BFS则是访问了最多的结点，重复路径较多； A*做过的路径数在BFS和DFS之间，并且在测试里，A *扩展了更少的结点就获得了和BFS一样的分数及cost

Multiagent

minimax

算法介绍

- minimax算法的前提是双方都是做最优决策

- 参考书上伪代码

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

  可知，算法的核心即递归调用：要求当前结点的极小极大值，则要求当前结点所有后继的极小极大值，并根据当前结点是MAX还是MIN来取后继的极小极大值中的max还是min

- 时间复杂度：$O(b^m)$，后继为b个，深度为m
- 空间复杂度：O(bm) 或 O(m)，取决于一次性生成所有后继还是每次生成一个后继

实验过程

- 实现的思路
  - 若为终止态，则返回效用值
  - 若当前为吃豆人并且depth=0，也返回当前状态和效用值，相当于depth限制了每次探索的深度，一个深度以pacman起始，到下一次轮到pacman之前。
  - 否则，则遍历所有子结点，若当前结点为pacman，即MAX结点，则取孩子节点里minimax值最大的那个结点及分数。若为ghost结点，即MIN结点，则取孩子结点里minimax值最小的那个结点及分数。
- 代码：

```
def minimax(self, state, depth):#depth是算法搜索的深度
        if state.isTerminated():
            return None, state.evaluateScore()
        if state.isMe() and depth == 0:#即算法已经不允许继续往下搜索了
            return state, state.evaluateScore()
        best_state, best_score = None, -float('inf') if state.isMe() else
float('inf')
        if state.isMe():
            depth = depth - 1
        for child in state.getChildren():
            # YOUR CODE HERE
            _,cur_score = self.minimax(child,depth)
            #对于pacman，要求max
            if state.isMe():
                if best_score<cur_score:
                    best_score=cur_score
                    best_state=child
```

```
            else:#对ghost，要min
                if best_score>cur_score:
                    best_score=cur_score
                    best_state=child
        return best_state, best_score
```

结果分析

- ```
  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
  ```
  ```

- ```
      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 0 seconds.
  *** Won 0 out of 1 games. Average score: 84.000000 ***
  *** PASS: test_cases\q2\8-pacman-game.test

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


  Finished at 22:29:54

  Provisional grades
  ==================
  Question q2: 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.
  ```

alpha-beta剪枝

算法介绍

- 在minimax算法的基础上，尽可能根据已知信息消除部分搜索树（即剪枝），即剪掉那些不可能影响决策的分支，在返回相同的结果的前提下减少搜索空间，提高效率
- alpha-beta剪枝的效率很大程度上依赖于检查后继状态的顺序
- 伪代码：
  ```

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

实验过程

- 代码如下

```python
class MyAlphaBetaAgent():

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

    def alphabeta_cut(self,state,depth,alpha,beta):
        if state.isTerminated():
            return None, state.evaluateScore()
        if state.isMe() and depth == 0:
            return state, state.evaluateScore()
        best_state, best_score = None, -float('inf') if state.isMe() else 
float('inf')

        if state.isMe():
            depth=depth-1
        for child in state.getChildren():
            _,cur_score = self.alphabeta_cut(child,depth,alpha,beta)
            if state.isMe():
                if cur_score>beta:
                    return child,cur_score
                alpha=max(cur_score,alpha)
                if best_score<cur_score:
                    best_score=cur_score
                    best_state=child
            else:
```

```
                if cur_score<alpha:
                    return child,cur_score
                beta=min(cur_score,beta)
                if best_score>cur_score:
                    best_score=cur_score
                    best_state=child
        return best_state,best_score
    def getNextState(self, state):
        # YOUR CODE HERE
        alpha=-float('inf')#当前为止发现的MAX的最佳值选择
        beta=float('inf')#当前为止发现的MIN结点的最佳选择
        best_state,best_score=self.alphabeta_cut(state, self.depth, alpha, beta)
        return best_state
```

- 这部分的关键即是要理解对当前节点的所有子结点的遍历逻辑
    - 对每个子结点调用alpha-beta剪枝算法获得其minimax值，即为value
        - 如果当前是pacman，并且该value>beta，那么需要剪枝，因为pacman的上一层的MIN节点肯定不会选它，而更倾向于选beta的值。所以pacman结点剩下的子结点也不必再探索。

          否则根据值来更新alpha和当前最优状态、效用值
        - 如果当前是Ghost，并且value<alpha, 那么需要剪枝，因为ghost的上一层的MAX节点肯定不会选它，而更倾向于选alpha的值。所以Ghost结点剩下的子结点也不必再探索

          否则根据值来更新beta和当前最优状态、效用值

结果分析

```
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).
```

- ```
      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 0 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 22:35:48

  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.
  ```

总结

- 通过本次实验，自己复习了5种算法的基本思路对于其理解更加深刻了，特别是alpha-beta剪枝算法，在课上学习之后当时可能没有完全理解透，再来实现代码，通过查询资料理解逻辑并实现代码后，有种醍醐灌顶的感觉
- 整体实验轻松而有效，很好的对课上所学知识进行了巩固复习，也起到了温故知新的作用。
  ```