

# 人工智能基础 LAB1

PB19030861 王湘峰

## 实验整体描述

本次实验有 2 个部分，分别是 Search 和 Multiagent。具体而言，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。抽象而言，Search 实现的静态查找算法，Multiagent 的问题是在有对手的情况下做出下一步决策使自己的利益最大化。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。

## 实验原理

### Problem1: BreadthFirstSearch

```
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)
```

### Problem2: AStarSearch

假设好  $h(n)$  是结点  $n$  到目标结点代价的估计值，令  $g(n)$  为从初始结点到结点  $n$  已经花费的代价。那么设  $f(n)$  满足：

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

此时  $f(n)$ =经过结点  $n$  的最小代价解的估计代价，同时可以证明  $f(n)$ 满足可采纳性和一致性。具体算法如下：

```
function AStar-Search(problem)returns a solution, or failure  
     $node \leftarrow$  a node with STATE=problem.INITIAL-STATE, PATH-COST=0,  $f(n)=h(n)$   
     $frontier \leftarrow$  a priority queue ordered by  $f(n)$ , with  $node$  as the only element  
     $explored \leftarrow$  an empty set  
    loop do  
        if EMPTY?( $frontier$ ) then return failure  
         $node \leftarrow$  POP( $frontier$ )  
        if problem.GOAL-TEST( $node$ .STATE) then return SOLUTION( $node$ )  
        for each  $action$  in problem.ACTIONS( $node$ .STATE) do  
             $child \leftarrow$  CHILD-NODE(problem,  $node$ ,  $action$ )  
            if  $child$ .STATE is not in  $explored$  or  $frontier$  then  
                 $frontier \leftarrow$  INSERT( $child$ ,  $frontier$ )  
            else if  $child$ .STATE is in  $frontier$  with higher  $f(n)$  then  
                replace that  $frontier$  node with  $child$ 
```

Problem3:

(1)Minimax

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

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
  return v
```

## (2) Alpha-beta Pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$ 
  return the action in ACTIONS(state) with value v
```

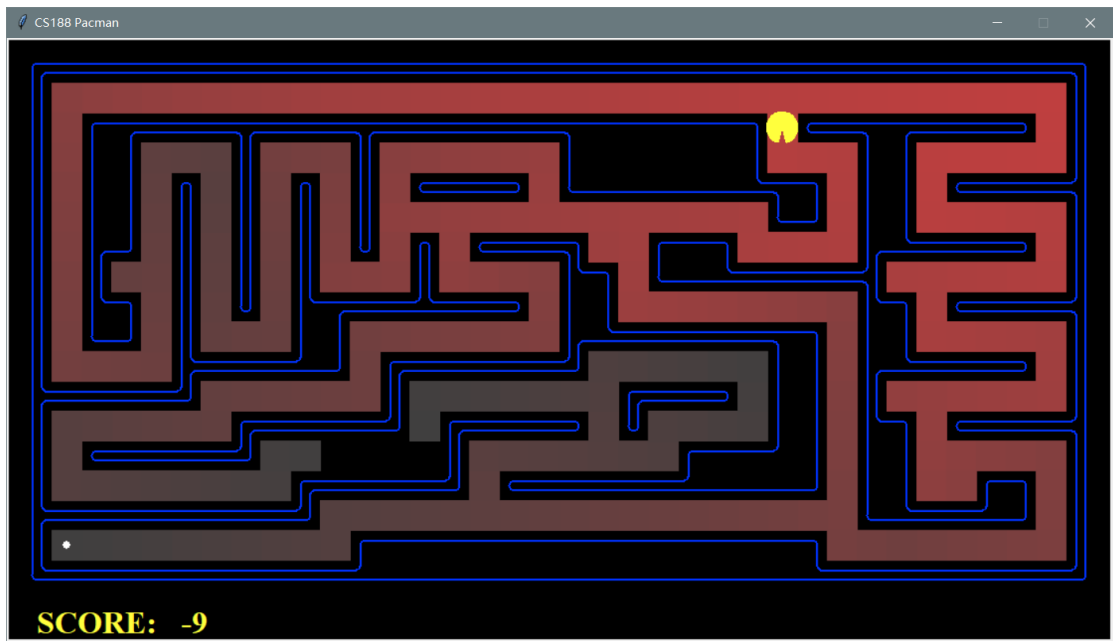
```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

## 实验过程与 debug 日志

Problem1（较为顺利）

只需要将栈的数据结构换成队列即可。



图为 BFS 运行的时的快照

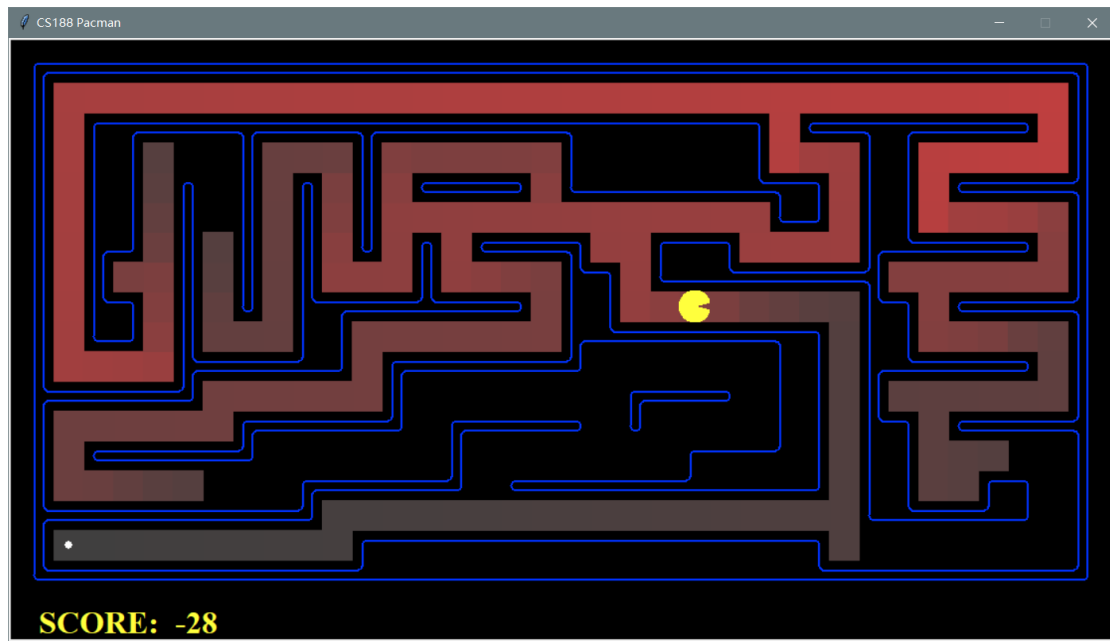
## Problem2

Try1:

错误的把初始结点的  $f(n)$  的值当成了 0，导致了失败。然后发现自己还没有完全理解算法，于是重新看书啃算法发现了问题。

Try2:

由于粗心把单步的耗散  $\text{step-cost}$  当做了从初始点到该点的总耗散值，导致出现了问题。在思考如何存储每个结点的  $g(n)$  时受到了示例的启发，采用字典  $\text{sumcost}$  存储总耗散，程序正常运行。



图为 AStar 算法运行时的快照

### Problem3

#### Minimax

#### Try1:

出现了扩展异常节点的问题。发现错误的把 `isTerminal()` 函数当成判断递归是否到原子状态的唯一条件，甚至没有考虑参数 `depth`。

#### Try2:

这次考虑了 `depth` 并加入 `if depth == 0` 作为终止条件，可是依然未能正确运行结果。经过研究发现只有当经历一回合之后（即所有 agent 都作出一步行动之后）`depth` 才会减 1。于是在递归中加入 `if child isMe()` 并只对为 `TRUE` 的情况将 `depth-1`。结果是 `passed` 的测试文件增加了许多，可是依然在部分测试中出现扩展异常结点的情况。

Try3:

经过仔细推敲，发现第一次出现  $\text{depth}=0$  时必然是 pacman 的 max 结点层，此时其他 min 节点 agent 尚未做出行动。因此必须想办法让其他 agent 作出行动后再终止程序。经过思考和修改，对  $\text{depth}=0$  的情况进行了更加详尽的分类，Minimax 函数成功运行！

### Alpha-beta Pruning

Try1:

经过 problem2 失败的经验，我首先把算法和原理部分仔仔细细看了个遍，确定了递归函数的参数（相较于 Minimax 增加了参数  $\alpha$  和  $\beta$ ），以及对  $\text{depth}=0$  的情况进行了考虑。此时发现函数的  $\alpha$  和  $\beta$  需要在函数“之外”进行初始化，于是我“自作主张”在 `__init__(self,depth)` 中加了 `self.alpha` 和 `self.beta` 的初始化。可是运行结果出人意料：所有的测试文件全部 pass，pacman 程序正常运行可是 test 却没有过。

```

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 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** FAIL: test_cases\q3\8-pacman-game.test
*** Bug: Wrong number of states expanded.
*** Tests failed.

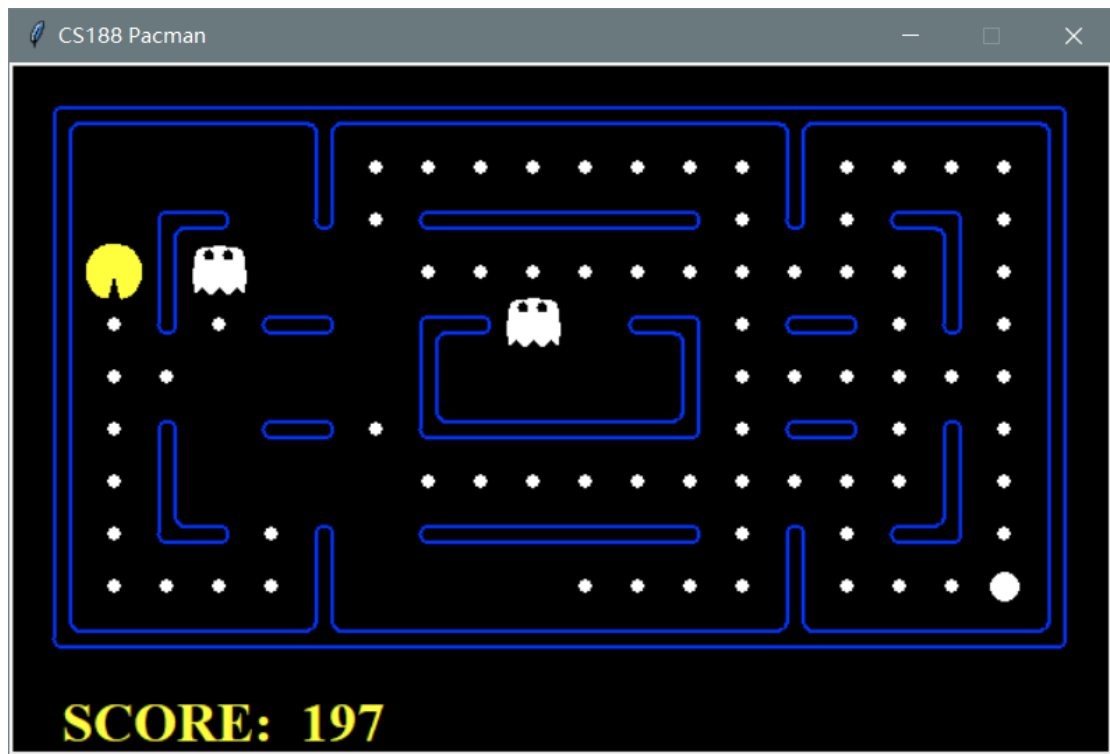
```

图为 test 结果

Try2:

经过一个多小时的推敲，终于发现了真正的原因：此前本人偷懒，在  $\text{depth}=0$  的时候，认为都已经到终端结点了，alpha-beta 对性能的提升并不算大，于是完全按照 Minimax 的做法将所有终端结点都评估了一遍。这样就导致了扩展了本应该 cut 掉的结点，导致了

test 报错，但同时程序能够正确运行。经过修改，最终全部程序完美运行！



图为 test3 运行时的快照

事后回想起来，为了偷一分钟敲代码的懒，却花费了一个小时来 debug，可谓捡了芝麻丢了西瓜。经过这次教训，以后再也不能随便偷懒了

## 实验结果分析

在 test1 和 test2 中，pacman 都找到了最优的路线。这说明 BFS 算法和 AStar 算法都是完备的，这与上课时推导的结论相符合。



```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 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
Starting on 5-14 at 11:40:32
```

图为 BFS 运行结果

```
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 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
Starting on 5-14 at 11:40:36
```

图为 AStar 运行结果

对于 test3，经过多次运行程序，实战结果平均在 1600 分左右：

```
Pacman emerges victorious! Score: 1451
Average Score: 1451.0
Scores:      1451.0
Win Rate:    1/1 (1.00)
Record:      Win
```

```
Pacman emerges victorious! Score: 1692
Average Score: 1692.0
Scores:      1692.0
Win Rate:    1/1 (1.00)
Record:      Win
```

```
Pacman emerges victorious! Score: 1878
Average Score: 1878.0
Scores:      1878.0
Win Rate:    1/1 (1.00)
Record:      Win
```

图为三次 test3 的结果

这个成绩与人类玩家不相上下，算法的效果是较为显著的。

但是在实验的过程中还观察到有些难以理解的行为，例如 ghost 离 pacman 较远的时候程序选择原地踏步等。初步分析是因为思考深度 depth 的限制，导致结点评估值与真实值差异较大。如果能够

将 depth 的值提高的话，得分的平均值应该会更近些，唯一的限制因素是单位时间内的算力。

## 实验总结

经过此次实验，更加直观的看到了不同搜索算法的空间复杂度等特点，并且对算法本身加深了印象与理解。例如三种算法中，空间复杂度排序为：BFS > AStar > DFS，但只有 BFS 和 AStar 是完备的。此外对于问题的形式化也有了一定的理解，通过对复杂的吃豆人问题抽象成为搜索问题，进而简化了工作的难度，这在以后解决实际问题时也是需要学习的。

除了实验本身的内容外，在完成实验的过程中也发现了自身的诸多不足，如粗心、耍小聪明、操之过急等。以后学习工作中应当注意避免。

总之本次实验是对前一阶段搜索算法学习效果的一个较好的检验，在实践中我也加深了对算法的理解。