

P01 Pacman Game

学号	姓名	专业(方向)
18340052	何 泽	计算机科学与技术 (超级计算方向)
18340032	邓俊锋	计算机科学与技术 (大数据与人工智能方向)

目录

1. Idea of A* Algorithm

2. Idea of Min-Max and alpha-beta pruning algorithms

3. Codes

[Question 1](#)

[Question 2](#)

[Question 3](#)

[Question 4](#)

[Question 5](#)

4. 结果展示

[Question 1](#)

[Question 2](#)

[Question 3](#)

[Question 4](#)

[Question 5](#)

5. 结果分析

[\(1\) Search in Pacman](#)

[\(2\) Multi-Agent Pacman](#)

6. Experimental experience

1. Idea of A* Algorithm

- A* 是一种启发式搜索，与传统的搜索相比，它对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这种方式可以省略大量无畏的搜索路径，提

高了效率。

2. Idea of Min-Max and alpha-beta pruning algorithms

- *Min – Max*

极小极大算法用于博弈树搜索，即游戏一方想要极大化最终结果，另外一方想要极小化最终结果，且两者交替执行策略。极小极大算法对博弈树执行完整的深度优先探索，递归算法自上而下一直前进到树的叶结点，然后随着递归回溯通过搜索树把极小极大值回传。若当前节点为极大方，则取子节点中最大的回传给父节点；反之回传最小的。

- $\alpha - \beta$ *pruning*

极小极大值搜索的问题是必须检查的游戏状态的数目是随着博弈的进行呈指数级增长，虽然无法让它变成多项式级，但是仍可以删掉很多不需要进行搜索的节点。

alpha – beta 剪枝可以应用于任何深度的树，很多情况下可以剪裁整个子树，而不仅仅是剪裁叶结点。一般原则是：考虑在树中某处的结点 n , 选手选择移动到该结点。如果选手在 n 的父结点或者更上层的任何选择点有更好的选择 m ，那么在实际的博弈中就永远不会到达 n 。所以一旦发现关于 n 的足够信息（通过检查它的某些后代），能够得到上述结论，我们就可以剪裁它，即

- 只要当前 Max 结点的值 \geq 祖先某一 Min 结点的值，就可以在该 Max 结点上做 α 剪枝
- 只要当前 Min 结点的值 \leq 祖先某一 Max 结点的值，就可以在该 Min 结点上做 β 剪枝

3. Codes

Question 1

```
1 #search.py
2
3 def breadthFirstSearch(problem):
4     """Search the shallowest nodes in the search tree first."""
5     q = util.Queue()
6     beg = problem.getStartState()
```

```

7     q.push((beg, []))
8     visited = []
9     while not q.isEmpty():
10        cur, actions = q.pop()
11        if cur in visited:
12            continue
13        visited.append(cur)
14        if problem.isGoalState(cur):
15            break
16        for suc in problem.getSuccessors(cur):
17            q.push((suc[0], actions + [suc[1]]))
18    return actions
19
20 def aStarSearch(problem, heuristic=nullHeuristic):
21     """Search the node that has the lowest combined cost and
heuristic first."""
22     q = util.PriorityQueue()
23     beg = problem.getStartState()
24     q.push((0, beg, []), 0)
25     visited = []
26     while not q.isEmpty():
27         cost, cur, actions = q.pop()
28         if cur in visited:
29             continue
30         visited.append(cur)
31         if problem.isGoalState(cur):
32             break
33         for suc in problem.getSuccessors(cur):
34             priority = cost + suc[2] + heuristic(suc[0], problem)
35             q.push((cost + suc[2], suc[0], actions + [suc[1]]),
priority)
36     return actions

```

Question 2

```

1 #searchAgents.py
2
3 class CornersProblem(search.SearchProblem):

```

```

4      """
5          This search problem finds paths through all four corners of a
6          layout.
7      """
8
9      def __init__(self, startingGameState):
10         """
11             Stores the walls, pacman's starting position and corners.
12         """
13         self.walls = startingGameState.getWalls()
14         self.startingPosition =
15             startingGameState.getPacmanPosition()
16             top, right = self.walls.height-2, self.walls.width-2
17             self.corners = ((1,1), (1,top), (right, 1), (right, top))
18             for corner in self.corners:
19                 if not startingGameState.hasFood(*corner):
20                     print 'Warning: no food in corner ' + str(corner)
21             self._expanded = 0 # DO NOT CHANGE; Number of search
22             nodes expanded
23             # Please add any code here which you would like to use
24             # in initializing the problem
25             self.startingGameState = startingGameState
26             visited = [0,0,0,0]
27             for (i,corner) in enumerate(self.corners):
28                 if self.startingPosition == corner:
29                     visited[i] = 1
30             self.startState = (self.startingPosition,visited)
31
32
33             def getStartState(self):
34                 """
35                     Returns the start state (in your state space, not the
36                     full Pacman state
37                     space)
38                 """
39
40                 return self.startState
41
42
43             def isGoalState(self, state):
44                 """
45

```

```

38         Returns whether this search state is a goal state of the
problem.
39         """
40         if state[1] == [1,1,1,1]:
41             return True
42         return False
43
44     def getSuccessors(self, state):
45         """
46             Returns successor states, the actions they require, and a
cost of 1.
47             As noted in search.py:
48                 For a given state, this should return a list of
triples, (successor,
49                         action, stepCost), where 'successor' is a successor
to the current
50                         state, 'action' is the action required to get there,
and 'stepCost'
51                         is the incremental cost of expanding to that
successor
52         """
53
54         successors = []
55         for action in [Directions.NORTH, Directions.SOUTH,
Directions.EAST, Directions.WEST]:
56             # Add a successor state to the successor list if the
action is legal
57             # Here's a code snippet for figuring out whether a
new position hits a wall:
58             #     x,y = currentPosition
59             #     dx, dy = Actions.directionToVector(action)
60             #     nextx, nexty = int(x + dx), int(y + dy)
61             #     hitsWall = self.walls[nextx][nexty]
62             x, y = state[0]
63             dx, dy = Actions.directionToVector(action)
64             nextx, nexty = int(x + dx), int(y + dy)
65             if not self.walls[nextx][nexty]:
66                 next = [item for item in state[1]]
67                 if (nextx,nexty) in self.corners:

```

```

68             i = self.corners.index((nextx,nexty))
69             next[i] = 1
70             nextState = ((nextx,nexty),next)
71             successors.append((nextState,action,1))
72             self._expanded += 1 # DO NOT CHANGE
73             return successors
74
75     def getCostOfActions(self, actions):
76         """
77             Returns the cost of a particular sequence of actions.  If
78             those actions
79                 include an illegal move, return 999999.  This is
80             implemented for you.
81             """
82             if actions == None: return 999999
83             x,y= self.startingPosition
84             for action in actions:
85                 dx, dy = Actions.directionToVector(action)
86                 x, y = int(x + dx), int(y + dy)
87                 if self.walls[x][y]: return 999999
88             return len(actions)
89
90
91     def cornersHeuristic(state, problem):
92         """
93             A heuristic for the CornersProblem that you defined.
94
95             state:  The current search state
96                     (a data structure you chose in your search
97             problem)
98
99             problem: The CornersProblem instance for this layout.
100
101            This function should always return a number that is a lower
102            bound on the
103            shortest path from the state to a goal of the problem; i.e.
104            it should be
105            admissible (as well as consistent).
106            """

```

Question 3

```
1 #searchAgents.py
2
3 def foodHeuristic(state, problem):
4     """
5         Your heuristic for the FoodSearchProblem goes here.
6
7         This heuristic must be consistent to ensure correctness.
8
9         First, try to come
10            up with an admissible heuristic; almost all admissible
11            heuristics will be
12            consistent as well.
13
14            If using A* ever finds a solution that is worse uniform cost
15            search finds,
16            your heuristic is *not* consistent, and probably not
17            admissible! On the
18            other hand, inadmissible or inconsistent heuristics may find
19            optimal
20            solutions, so be careful.
```

```

16     The state is a tuple ( pacmanPosition, foodGrid ) where
17     foodGrid is a Grid
18         (see game.py) of either True or False. You can call
19     foodGrid.asList() to get
20         a list of food coordinates instead.
21
22     If you want access to info like walls, capsules, etc., you can
23     query the
24     problem. For example, problem.walls gives you a Grid of where
25     the walls
26     are.
27
28     If you want to *store* information to be reused in other calls
29     to the
30     heuristic, there is a dictionary called problem.heuristicInfo
31     that you can
32     use. For example, if you only want to count the walls once and
33     store that
34     value, try: problem.heuristicInfo['wallCount'] =
35     problem.walls.count()
36     Subsequent calls to this heuristic can access
37     problem.heuristicInfo['wallCount']
38     """
39
40     position, foodGrid = state
41     foodLst = foodGrid.asList()
42
43
44     if len(foodLst) == 0:
45         return 0
46     ans = max(map(lambda
47         x:mazeDistance(position,x,problem.startingGameState), foodLst))
48
49     return ans

```

Question 4

```

1 class MinimaxAgent(MultiAgentSearchAgent):
2     """
3         Your minimax agent (question 2)
4         """

```

```

5
6     def getAction(self, gameState):
7
8         def max_agent(state, depth):
9             if state.isWin() or state.isLose() or depth == self.depth:
10                 return self.evaluationFunction(state), None
11             score = float('-Inf')
12             max_action = ''
13             for action in state.getLegalActions(0):
14                 tmp_score = min_agent(state.generateSuccessor(0, action), depth, 1)[0]
15                 if score < tmp_score:
16                     score = tmp_score
17                     max_action = action
18             return score, max_action
19
20         def min_agent(state, depth, ghostNum):
21             if state.isWin() or state.isLose():
22                 return self.evaluationFunction(state), None
23             score = float('Inf')
24             min_action = ''
25             if ghostNum == gameState.getNumAgents() - 1:
26                 for action in state.getLegalActions(ghostNum):
27                     tmp_score =
28                         max_agent(state.generateSuccessor(ghostNum, action), depth + 1)[0]
29                     if score > tmp_score:
30                         score = tmp_score
31                         min_action = action
32             else:
33                 for action in state.getLegalActions(ghostNum):
34                     tmp_score =
35                         min_agent(state.generateSuccessor(ghostNum, action), depth,
36                         ghostNum + 1)[0]
37                     if score > tmp_score:
38                         score = tmp_score
39                         min_action = action
40
41             return score, min_action

```

Question 5

```
1 class AlphaBetaAgent(MultiAgentSearchAgent):
2     """
3         Your minimax agent with alpha-beta pruning (question 3)
4     """
5
6     def getAction(self, gameState):
7         """
8             Returns the minimax action using self.depth and
9             self.evaluationFunction
10            """
11            def max_agent(state, depth, alpha, beta):
12                if state.isWin() or state.isLose() or depth == self.depth:
13                    return self.evaluationFunction(state), None
14                score = float('-Inf')
15                max_action = ''
16                for action in state.getLegalActions(0):
17                    tmp_score = min_agent(state.generateSuccessor(0,
18 action), depth, 1, alpha, beta)[0]
19                    if score < tmp_score:
20                        score = tmp_score
21                        max_action = action
22                    if score > beta:
23                        return score, max_action
24                    alpha = max(alpha, score)
25                return score, max_action
26
27            def min_agent(state, depth, ghostNum, alpha, beta):
28                if state.isWin() or state.isLose():
29                    return self.evaluationFunction(state), None
30                score = float('Inf')
31                min_action = ''
```

```
30         # Last ghost, go to next max ply
31         if ghostNum == gameState.getNumAgents() - 1:
32             for action in state.getLegalActions(ghostNum):
33                 tmp_score =
34                     max_agent(state.generateSuccessor(ghostNum, action), depth + 1,
35 alpha, beta)[0]
36                     if score > tmp_score:
37                         score = tmp_score
38                         min_action = action
39                     if score < alpha:
40                         return score, min_action
41                         beta = min(beta, score)
42                     else:
43                         for action in state.getLegalActions(ghostNum):
44                             tmp_score =
45                             min_agent(state.generateSuccessor(ghostNum, action), depth,
46 ghostNum + 1, alpha, beta)[0]
47                             if score > tmp_score:
48                                 score = tmp_score
49                                 min_action = action
50                             if score < alpha:
51                                 return score, min_action
52                                 beta = min(beta, score)
53             return score, min_action
54
55     maxiscore, bestAction = max_agent(gameState, 0, float('-
56 Inf'), float('Inf'))
57     return bestAction
58     util.raiseNotDefined()
```

4.结果展示

Question 1

```
heze@HeZes-MacBook-Pro: ~/Desktop/P01_Pacman/search python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores: 300.0
Win Rate: 1/1 (1.00)
Record: Win
```

Question 2

```
heze@HeZes-MacBook-Pro: ~/Desktop/P01_Pacman/search python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
Path found with total cost of 106 in 8.7 seconds
Search nodes expanded: 801
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores: 434.0
Win Rate: 1/1 (1.00)
Record: Win
```

- 共搜索 434 个节点

Question 3

```
heze@HeZes-MacBook-Pro: ~/Desktop/P01_Pacman/search python pacman.py -l trickySearch -p AStarFoodSearchAgent
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
Path found with total cost of 60 in 32.6 seconds
Search nodes expanded: 4137
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores: 570.0
Win Rate: 1/1 (1.00)
Record: Win
```

- 共搜索 4137 个节点

Question 4

```
heze@HeZes-MacBook-Pro: ~/Desktop/P01_Pacman/multiagent python autograder.py -q q2
Starting on 9-28 at 16:56:30

Question q2
=====

*** PASS: test_cases/q2/0-lecture-6-tree.test
*** PASS: test_cases/q2/0-small-tree.test
*** PASS: test_cases/q2/1-1-minimax.test
*** PASS: test_cases/q2/1-2-minimax.test
*** PASS: test_cases/q2/1-3-minimax.test
*** PASS: test_cases/q2/1-4-minimax.test
*** PASS: test_cases/q2/1-5-minimax.test
*** PASS: test_cases/q2/1-6-minimax.test
*** PASS: test_cases/q2/1-7-minimax.test
*** PASS: test_cases/q2/1-8-minimax.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).
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
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 53 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 16:57:24

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.
```

- 通过所有测试

Question 5

```
heze@HeZes-MacBook-Pro: ~/Desktop/P01_Pacman/multiagent python autograder.py -q q3
Starting on 9-28 at 16:58:41

Question q3
=====

*** PASS: test_cases/q3/0-lecture-6-tree.test
*** PASS: test_cases/q3/0-small-tree.test
*** PASS: test_cases/q3/1-1-minimax.test
*** PASS: test_cases/q3/1-2-minimax.test
*** PASS: test_cases/q3/1-3-minimax.test
*** PASS: test_cases/q3/1-4-minimax.test
*** PASS: test_cases/q3/1-5-minimax.test
*** PASS: test_cases/q3/1-6-minimax.test
*** PASS: test_cases/q3/1-7-minimax.test
*** PASS: test_cases/q3/1-8-minimax.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).
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress this warning.
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 53 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 16:59:35

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.
```

- 通过所有测试

5.结果分析

(1) Search in Pacman

- 在 **Question 1** 里，启发式函数是离目标点的曼哈顿距离。启发函数十分简单，离目标越近，函数值越小；到达终点后启发式函数为0而；且比实际需要的cost小，所以能引导吃豆人走到目标点。
- 在 **Question 2** 里，我的启发式函数参考了 **Question 1** 的启发式函数。我的启发式函数是距离未访问目标点 (Corners) 迷宫距离 (在 `search.py` 中用 `bfs` 实现) 的最大值。每到达一个角落，启发式函数都会减少；到达终点后启发式函数为0；而且比实际需要的cost小，所以能引导吃豆人走完四个角落。虽然计算启发函数很慢，但是能大大减小搜索空间。
- 在 **Question 3** 里，启发式函数与 **Question 2** 类似。我的启发式函数是距离未吃的点的迷宫距离 (在 `search.py` 中用 `bfs` 实现) 的最大值。每吃掉了一个点，启发式函数都会减少；到达终点后启发式函数为0；而且比实际需要的cost小。所以能引导吃豆人吃完全部点。

(2) Multi-Agent Pacman

可以看出使用 $\alpha - \beta$ 剪枝之后在运行相同深度的时候更快，比如分别运行如下2条命令：

```
1 | python pacman.py -p MinimaxAgent -a depth=4 -l smallClassic
2 | python pacman.py -p AlphaBetaAgent -a depth=4 -l smallClassic
```

在运行的时候 **MinimaxAgent** 差不多是2~3秒移动一次，而 **AlphaBetaAgent** 差不多1秒就移动一次，可以明显感受到使用 **AlphaBetaAgent** 之后每一次移动需要计算的时间更短，移动过程表现地更为“流畅”，这便是剪枝的作用。

6.Experimental experience

这是人工智能的第一次项目作业，完成起来和之前的感觉完全不同，因为之前的都是一个文件即可完成，但是这次的项目有好多文件，这就要考虑文件之间的逻辑关系，在完成之前需要提前了解整个项目的代码框架以及涉及到的文件之间的关系。

除此之外，在 **Multi-Agent Pacman** 中，虽然完成了实验任务，但是在实际运行过程中还是可以看出很多问题的，比如经常停在某一个位置不动，直到怪物靠近的时候才会继续走，即使在旁边就有食物还没有怪物这种情况下有时也不会动；此外基本上每次都会输，都会被怪物吃掉。这些都代表现在的算法还是有一些缺陷的，可目前的算法已经是费了很大的力才写出来的，可见想要实现一个完美的算法是多么的难。

总之，通过自己的实现，这一项目加深了我对搜索、 $Min - Max$ 以及剪枝算法的理解。