# P01 Pacman Game

| 学号 | 姓名 | 专业(方向) |
|---|---|---|
| 18340215 | 张天祎 | 计科（大数据和人工智能） |
| 18340208 | 张洪宾 | 计科（超算） |

## 1.Idea of A* Algorithm (Use a few sentences to describe your understanding of the algorithm)

- 类似一致代价搜索，只是在拓展结点时使用g+h最小的，而不是g最小的。$A*$算法一方面参考当前已知的信息，一方面参考启发式函数给出的信息来拓展结点，在某种程度上类似人类的思维过程。但启发式函数的选取对$A*$影响很大，选取一个好的启发式函数不是一件容易的事情。

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

- 自己作最好选择的同时，也站在敌人的角度思考问题，敌人也作最好选择。树的各层形成交替的Min-Max决策树。
- 在自己或敌人有较好的选择时，一定不会选较差的分支。因此，当确定当前分支的结果一定比已遍历过的分支差时，就不用遍历当前分支的剩余部分。

## 3. Codes

**Question 1**

```
def aStarSearch(problem, heuristic=nullHeuristic):
    def f(node):
        return node[2] + heuristic(node[0],problem)
    frontier = util.PriorityQueueWithFunction(f)
    frontier.push((problem.getStartState(),[],0))
    rst = set()
    while not frontier.isEmpty():
        curr_sta,curr_act,curr_cost = frontier.pop()
        if problem.isGoalState(curr_sta):
            return curr_act
        if curr_sta not in rst:
            rst.add(curr_sta)
            for succ in problem.getSuccessors(curr_sta):
                succ_sta, succ_act, succ_cost = succ
                new_act = curr_act + [succ_act]
                new_cost = curr_cost + succ_cost
                frontier.push((succ_sta, new_act, new_cost))
    return []
```

**Question 2**

```
class CornersProblem(search.SearchProblem):
    def getStartState(self):
        return self.startingPosition, (0,0,0,0)
    def isGoalState(self, state):
        return state[1]==(1,1,1,1)
    def getSuccessors(self, state):
        successors = []
        for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
Directions.WEST]:
        x,y = state[0]
        dx, dy = Actions.directionToVector(action)
        nextx, nexty = int(x + dx), int(y + dy)
        hitsWall = self.walls[nextx][nexty]
        if not hitsWall:
            succ_pos = (nextx,nexty)
            succ_cnt = state[1]
            for i in range(4):
                if self.corners[i] == succ_pos:
                    tem = list(succ_cnt)
                    tem[i] = 1
                    succ_cnt = tuple(tem)
                successors.append(((succ_pos,succ_cnt),action,1))
        self._expanded += 1 # DO NOT CHANGE
        return successors
    def cornersHeuristic(state, problem):
        corners = problem.corners # These are the corner coordinates
        walls = problem.walls # These are the walls of the maze, as a Grid
(game.py)
        sum = 0
        x,y = state[0]
        goal = []
        for i in range(4):
            if not state[1][i]:
                goal.append(problem.corners[i])
        for corner in goal:
            min = 99999
            dis = abs(x-corner[0])+abs(y-corner[1])
        if min > dis:
            min = dis
            nx,ny = corner
        sum += min
        x,y = nx,ny
        goal.remove((x,y))
        return sum
```

### Question 3

```
def foodHeuristic(state, problem):
    position, foodGrid = state
    food = foodGrid.asList()
    pos_to_food = []
    food_to_food = []
    sum = 0
    for i in food:
        pos_to_food.append(abs(position[0] - i[0]) + abs(position[1] - i[1]))
        for j in food:
            food_to_food.append((abs(j[0] - i[0]) + abs(j[1] - i[1])))
    sum += min(pos_to_food) if pos_to_food else 0
    sum += max(food_to_food) if food_to_food else 0
    return sum
```

## Question 4

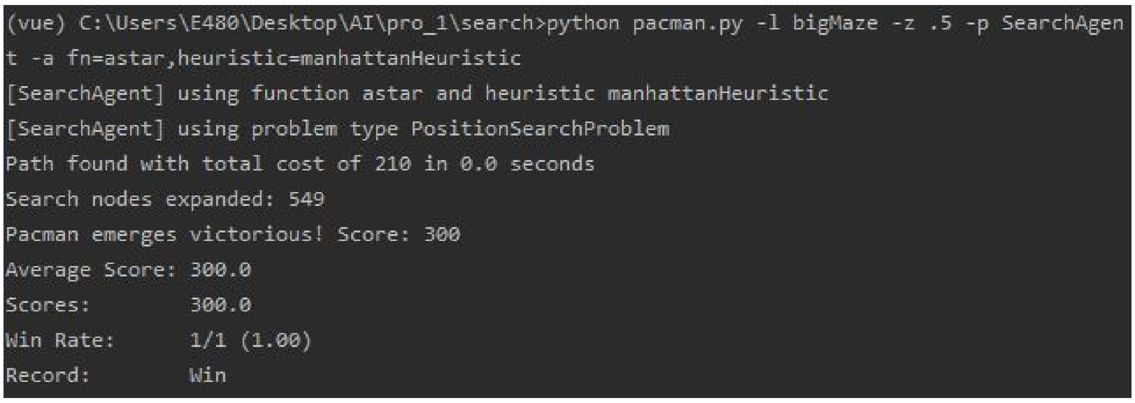
```python
def getAction(self, gameState):
    return self.MinMaxDFS(gameState, 1, 0)
def MinMaxDFS(self, gameState, currDepth, agentIndex):
    if agentIndex >= gameState.getNumAgents():  # 向下，agent变为0
        return self.MinMaxDFS(gameState, currDepth + 1, 0)
    if currDepth > self.depth or gameState.isWin() or gameState.isLose(): #终止
        return self.evaluationFunction(gameState)
    succ = gameState.getLegalActions(agentIndex)
    for i in succ:
        if i == 'Stop':
            succ.remove(i)
     scores = []
     for i in succ:
        scores.append(self.MinMaxDFS(gameState.generateSuccessor(agentIndex,
i),currDepth,agentIndex+1))
     if not agentIndex:
        mmax = max(scores)
        if currDepth == 1:
            bestIndices = []
            for i in range(len(scores)):
                if scores[i] == mmax:
                    bestIndices.append(i)
            return succ[random.choice(bestIndices)] # 伪随机一个最好的
        return mmax
      else:
            return min(scores)
```

## Question 5

```python
def getAction(self, gameState):
    return self.AlphaBetaDFS(gameState,1,0,-99999,99999)
def AlphaBetaDFS(self, gameState, currDepth, agentIndex,MIN,MAX):
    if agentIndex >= gameState.getNumAgents():  # 向下，agent变为0
        return self.AlphaBetaDFS(gameState, currDepth + 1, 0,MIN,MAX)
    if currDepth > self.depth or gameState.isWin() or gameState.isLose(): #终止
        return self.evaluationFunction(gameState)
    succ = gameState.getLegalActions(agentIndex)
    for i in succ:
        if i == 'Stop':
            succ.remove(i)
    if not agentIndex:
        if currDepth == 1:
            scores = []
            for i in succ:
scores.append(self.AlphaBetaDFS(gameState.generateSuccessor(agentIndex,i),currDe
pth,agentIndex+1,MIN,MAX))
            MMAX = max(scores)
            bestIndices = []
            for i in range(len(scores)):
                if scores[i] == MMAX:
                    bestIndices.append(i)
            return succ[random.choice(bestIndices)]
        MMAX = -99999
        for i in succ:
            MMAX = max(MMAX,self.AlphaBetaDFS(gameState.generateSuccessor(agentIndex,i),currDepth,agentIndex+1,MIN,MAX))
            if MAX <= MMAX: #剪枝
                return MMAX
            MAX = max(MAX,MMAX)
        return MMAX
    else:
        MMIN = 99999
        for i in succ:
            MMIN = min(MMIN,self.AlphaBetaDFS(gameState.generateSuccessor(agentIndex,i),currDepth,agentIndex+1,MIN,MAX))
            if MIN >= MMIN: #剪枝
                return MMIN
            MAX = min(MAX,MMIN)
        return MMIN
```

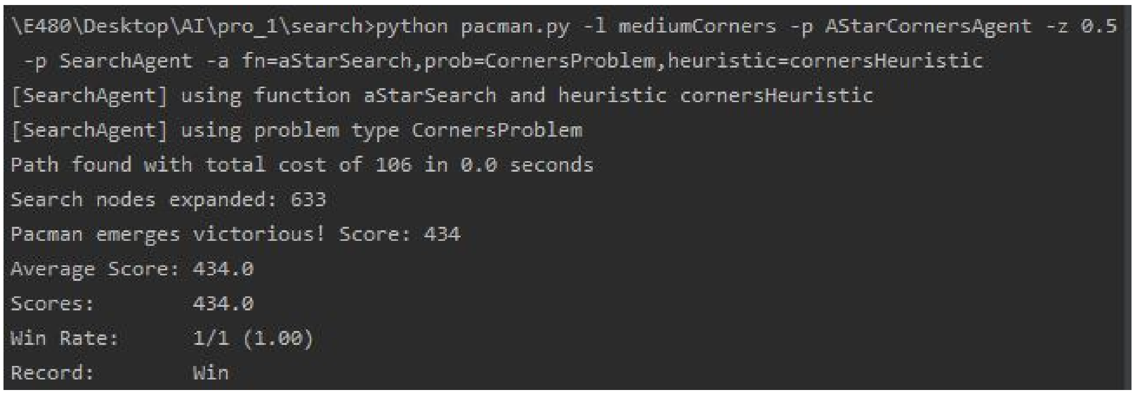
## 4.结果展示

- 1



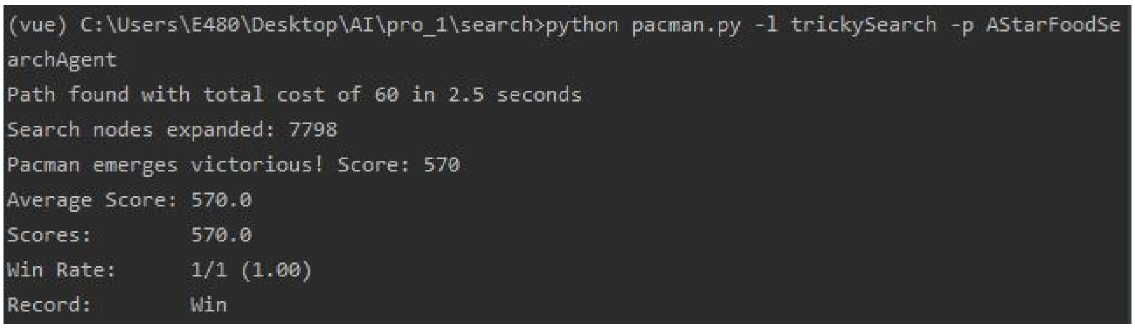
```
(vue) C:\Users\E480\Desktop\AI\pro_1\search>python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 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
```

- 2



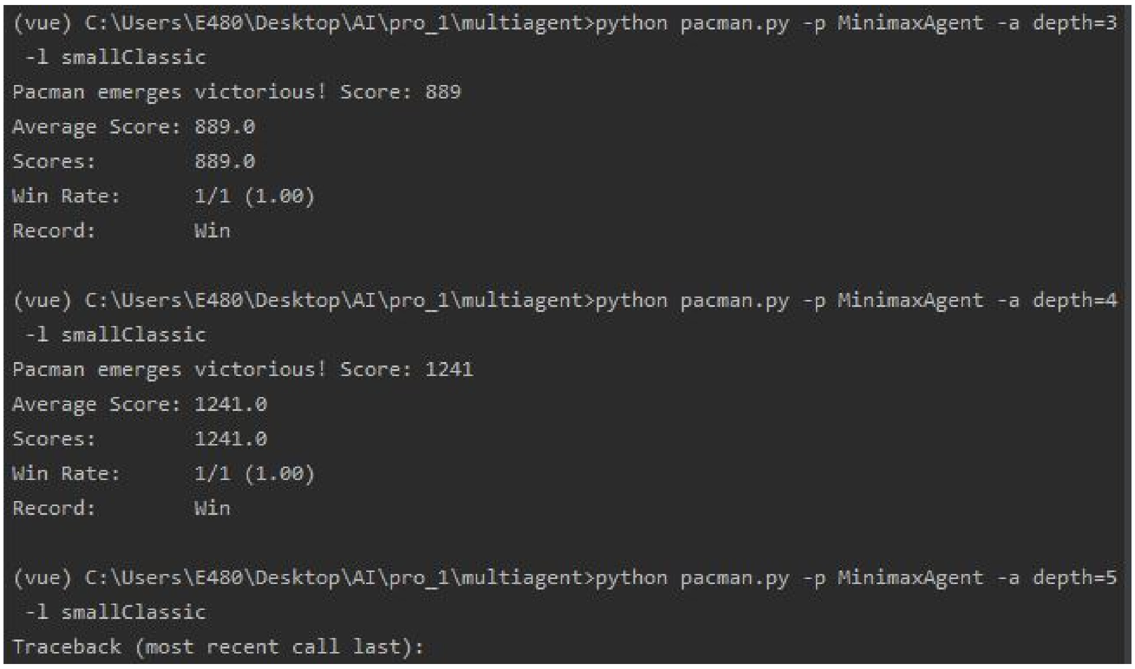
```
\E480\Desktop\AI\pro_1\search>python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
 -p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic
[SearchAgent] using function aStarSearch and heuristic cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 633
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

- 3



```
(vue) C:\Users\E480\Desktop\AI\pro_1\search>python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 2.5 seconds
Search nodes expanded: 7798
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

- 4

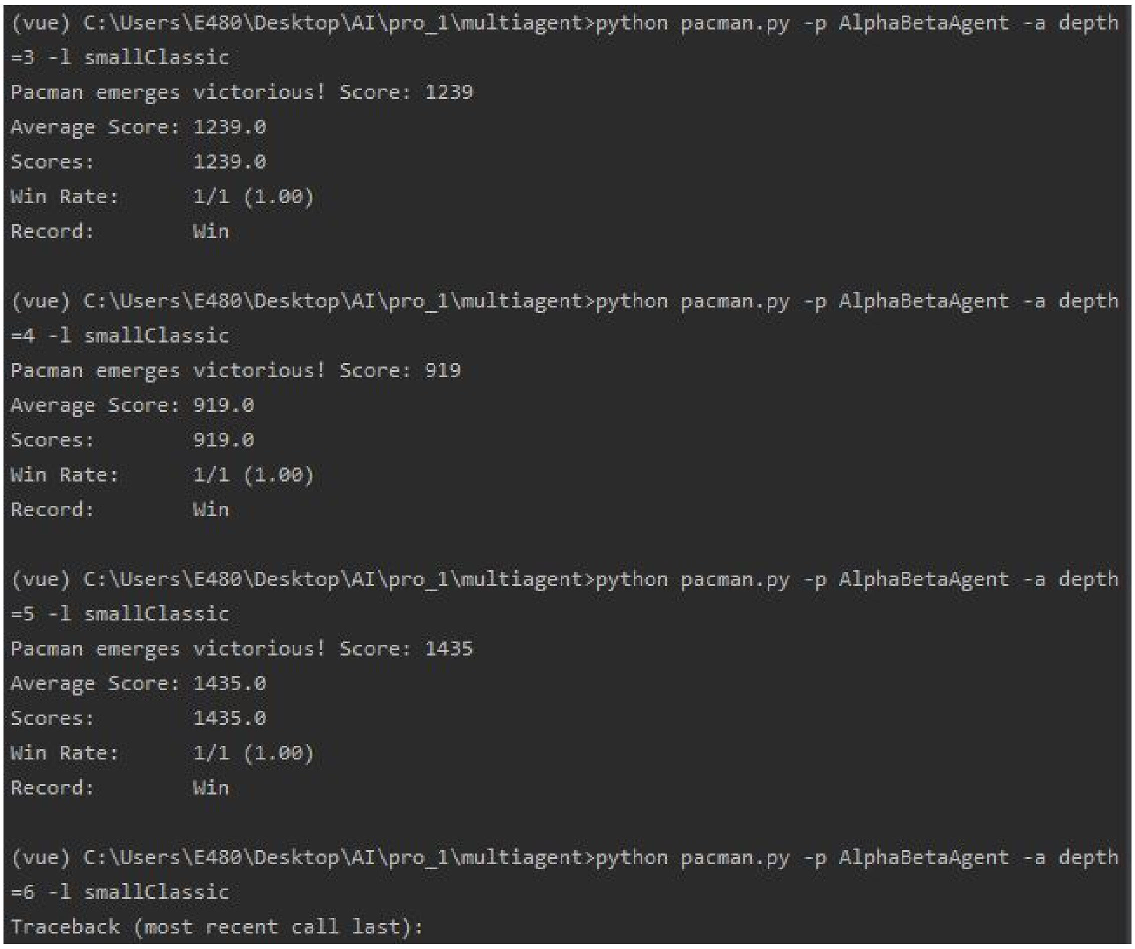


```
(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p MinimaxAgent -a depth=3
 -l smallClassic
Pacman emerges victorious! Score: 889
Average Score: 889.0
Scores:        889.0
Win Rate:      1/1 (1.00)
Record:        Win

(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p MinimaxAgent -a depth=4
 -l smallClassic
Pacman emerges victorious! Score: 1241
Average Score: 1241.0
Scores:        1241.0
Win Rate:      1/1 (1.00)
Record:        Win

(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p MinimaxAgent -a depth=5
 -l smallClassic
Traceback (most recent call last):
```

- 5



```
(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p AlphaBetaAgent -a depth
=3 -l smallClassic
Pacman emerges victorious! Score: 1239
Average Score: 1239.0
Scores:        1239.0
Win Rate:      1/1 (1.00)
Record:        Win

(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p AlphaBetaAgent -a depth
=4 -l smallClassic
Pacman emerges victorious! Score: 919
Average Score: 919.0
Scores:        919.0
Win Rate:      1/1 (1.00)
Record:        Win

(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p AlphaBetaAgent -a depth
=5 -l smallClassic
Pacman emerges victorious! Score: 1435
Average Score: 1435.0
Scores:        1435.0
Win Rate:      1/1 (1.00)
Record:        Win

(vue) C:\Users\E480\Desktop\AI\pro_1\multiagent>python pacman.py -p AlphaBetaAgent -a depth
=6 -l smallClassic
Traceback (most recent call last):
```

## 5.结果分析

### 1.Search in Pacman

- 第一问已经给出的启发式函数是当前位置到豆子的曼哈顿距离，是经典的启发式函数，考虑松弛问题，即忽视墙。显然是满足一致性的，则有是最优的。
- 第二问我们自己写的启发式函数的思路是使用贪心策略，依次去找曼哈顿距离最近的豆子。首先找最近的豆子，然后更新位置为该豆子处，再找下一个。其可满足性和一致性都没有证明出来。事实

上，我认为在特定的位置（地图的边界或中间附近）上该函数的两个性质可能都不能满足。但我们也找不出更好的函数了。在此问中由于墙的限制和初始位置因素，它碰巧得到了最优解。

- 第三问我们从上一问中获取类似的思路，把每一个豆子当作上一问的角落，使用贪心策略，结果拓展的节点为 9000+。不够理想，于是考虑新的方法。新的启发式函数有两个部分，一个是当前位置和最近的豆子的距离，第二个是所有豆子彼此之间的最大距离，然后对他们求和并返回。这样的距离肯定是实际距离的一个下界，是可满足的。但该函数似乎不具有一致性。
- 希望助教在讲解此问时能重点讲一下启发式函数的选取。

### 2.Multi-Agent Pacman

- 在使用Min-Max算法时，当搜索深度为3时，运行还算流畅，有一点卡顿，每步行动大概1s左右；当搜索深度为4时，运行已经很卡顿了，每一步行动大约2-3秒钟；当搜索深度为5时，基本很难正常运行了，每一步要等7-12s，由于我失去了耐性，使用Ctrl+C结束了程序。
- 在使用$\alpha-\beta$剪枝算法时，当搜索深度为3时，运行很流畅；当搜索深度为4时，运行有点卡顿，每步行动大概1s左右；当搜索深度为5时，卡顿明显，每步行动3-5s；当搜索深度为6时，运行困难，每一步要等15s左右。
- 两种算法运行到后期豆子很少时，似乎都会变慢一些。
- $\alpha-\beta$剪枝算法的优势显而易见。

## 6.Experimental experience

本次实验熟悉了A*算法，体会到了启发式函数选取的困境，也实现了Min-Max算法、$\alpha-\beta$剪枝算法。这次实验有几个较大的困难：

- 首先是是python，我们过去都只自学习了面向过程的python部分，面向对象的部分没有系统学习，这次实验前速成了一下。
- 其次代码阅读，每个问题都要先熟悉相关的代码接口函数，结构等，好在只要善用print，还是能快速上手。
- 最后是启发式函数的选取，深切地体会到了启发式函数是个玄学问题，实在是需要天赋和直觉，在问题2,3中都写过几个版本的启发式函数，但有的效果好，但不满足最优性条件，有的满足条件却又效果不好。但最后完成实验还是感觉即使是写出不理想的启发式函数，时间也没有白费，学到了很多。