P01 Pacman Game

学号	姓名	专业(方向)
18340161	田蕊	计算机科学与技术（超算方向）
18340149	孙新梦	计算机科学与技术（超算方向）

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

- 在DFS，BFS，一致代价，深度受限，迭代加深，以及双向搜索等**无信息搜索算法**中，是没有考虑过哪一个节点**更容易到达目标**的
- 在**启发式搜索**我们就会通过对节点的信息进行分析，估计他离目标节点的距离从而对要扩展的节点有所偏好，也就是在边界上用评估值来排序，评估值更小的更加优先被扩展
- 而**启发式搜索**的代表算法为A*搜索和IDA*搜索算法，我们在这个模块着重描述一下A*算法的**思路**：

  > 定义一个**评估函数**
  >
  > $$f(n) = g(n) + h(n)$$
  >
  > 其中
  >
  > - g(n)为到节点n的cost
  > - h(n)为从节点n到目标状态的cost的启发式估计值
  >
  > 这样得到的f(n)能够很好表示通过节点n走到目标状态的估计cost，我们用f(n)来给边界的节点排序，优先扩展节点f(n)值最小的那一个。

  ```
  A*算法伪函数：
  将起始点加入frontier
      当frontier不为空时：
        寻找frontier中f值最小的点current
        它是终止点，则找到结果，程序结束。
        否则，frontier移除visited，对current的每一个临近点
            若它不可走或在visited中，skip
            若它不在frontier中，加入。
            若它在frontier中，计算g值，若g值更小，替换其父节点为current，更新它的g值。
      若frontier为空，则路径不存在。
  ```

- 对于启发式搜索来说，**启发式函数**的优劣决定了扩展结点的数目。启发式函数需要具备
  - **可采纳性**：h(n)<=h*(n)，后者为最优路径的cost
  - **一致性**：对于任意两个节点n1和n2，有h(n1) ≤ c(n1 ->n2) + h(n2)
- 若有一致性，我们可以保证环检测能保留**最优性**，而若只有可采纳性，环检测可能不能保证最优性。
- 对于**问题1.3**，我们使用的启发式函数是题目代码给出的默认启发式函数，总是返回0，主要要求的是实现A*算法的代码，思路如下：

  > 首先用getStartState函数得到初始状态

> 创建knownstates列表，声明优先级队列PQ实例states
>
> 首先push进去初始状态，经过的动作序列为空，启发式函数值
>
> 当PQ不为空的时候，获取队首元素，也就是f=g+h值最小的那一个状态
>
> 若为终止状态，则返回经过的动作序列
>
> 若不在knownstate列表，则扩展进去作为已知，寻找状 态的后继，对于每一个后继获取坐标和方向，
>
> 若坐标不在knownstate里面，则组合出来当前动作， 计算节点f值，并在state中push进去改节点
>
> 最后返回动作序列

- 对于**问题1.4**，我们对于角落问题实现一个启发式函数评估当前状态到目标的距离，替代上一问的默认启发式函数
  - 角落问题就是四个角各有一个豆子，怎样在迷宫中吃掉四颗豆子，并有最优解
  - 在下面的结果分析中，我们会详细解读不同启发式函数的实现以及对比不同的性能。

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

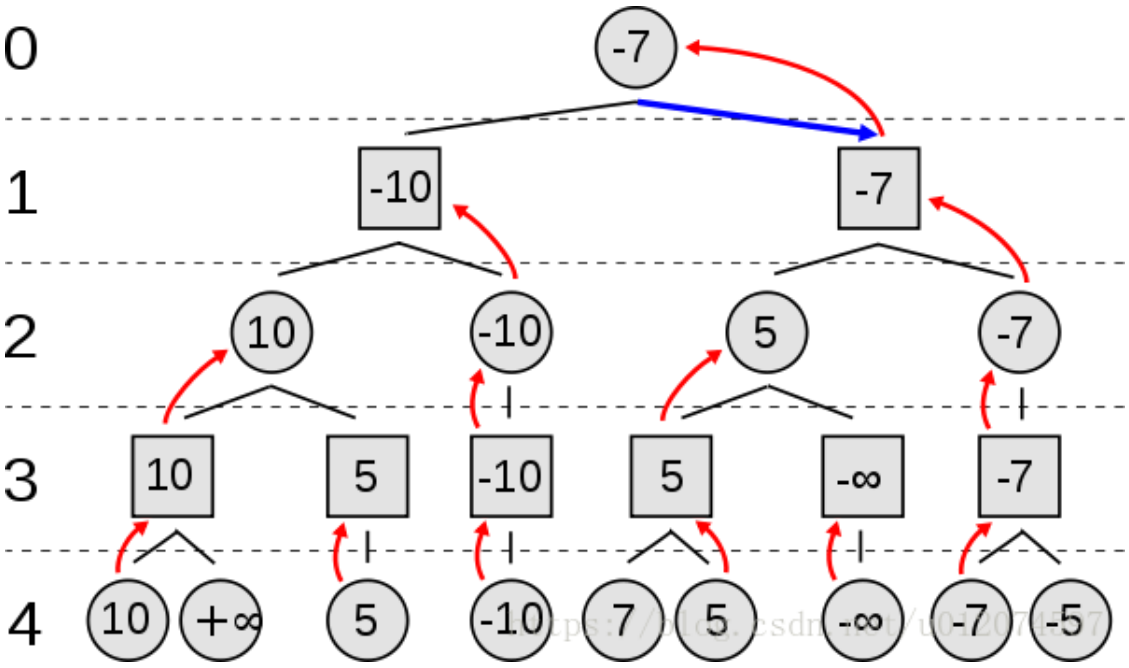
- ### The principle of Min-Max algorithm

  Min-Max算法是博弈树搜索的核心算法，在这个算法中我们有几个基本假设，分别是：

  - 零和游戏
  - 游戏只有有限多步
  - 游戏的状态或者决策可以映射到数值上
  - 每一步可以到达的后继都是确定的
  - 每一方的选手在每一步都是尽最大努力的

  在满足上述假设的情况下我们用Min-Max算法去做出每一步的决策。Min-Max算法的自然想法是将格局组织成一颗树，树的每一个节点表示一种格局，我们认为己方是极大方，对方是极小方。极大方选择自己的子节点中的最大值，而极小方选择自己的子节点中的最小值。在算法的开始，从当前状态开始进行深度优先搜索，直到到达终止状态或到达我们当前问题所能考虑的最大深度，根据极大方或极小方选择后继的得分来作为当前状态的评分。

  一个简单清晰的算法原理图如下所示，其中◯为极大方，▢为极小方：



总结一下Minimax算法的步骤：

1. 首先确定最大搜索深度D，D可能达到终局，也可能是一个中间格局。
2. 在最大深度为D的格局树叶子节点上，使用预定义的价值评价函数对叶子节点价值进行评价。
3. 自底向上为非叶子节点赋值。其中max节点取子节点最大值，min节点取子节点最小值。
4. 每次轮到我方时（此时必处在格局树的某个max节点），选择价值等于此max节点价值的那个子节点路径。

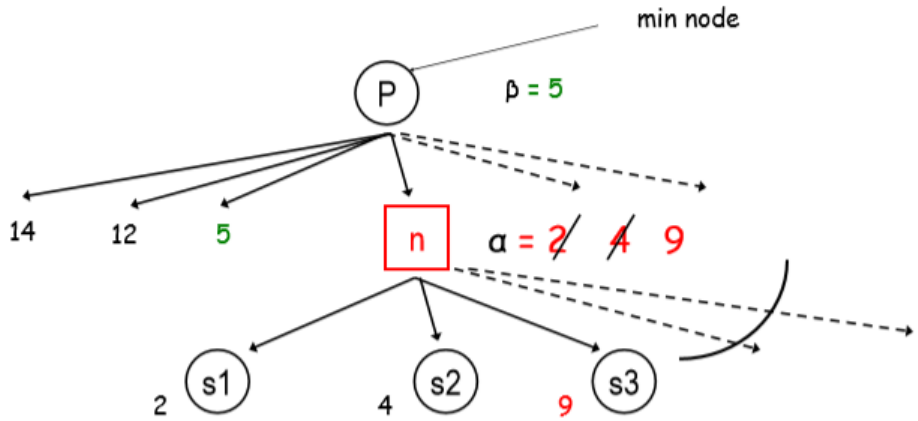
下面我们用伪代码的形式给出Min-Max算法一个更加清晰的描述：

```
DFMiniMax(n, Player)
if n is TERMINAL:
    return V(n)
ChildList = n.successors(Player)
if Player==Min
    return minimum of DFMiniMax(c,Max) over c ∈ ChildList
else:
    return maximum of DFMiniMax(c,Max) over c ∈ ChildList
```

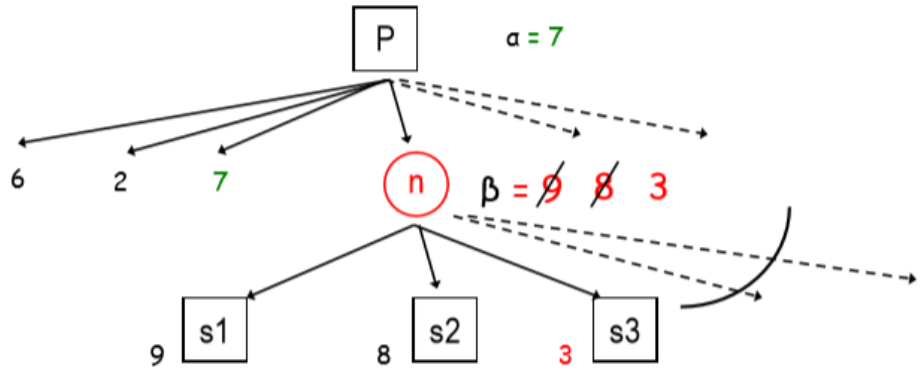
- **The principle of α-β pruning**

α-β pruning实际上是进行两种剪枝一种是对极大节点进行剪枝，也就是我们所说的α pruning，另一种是对极小节点进行剪枝，也就是我们所说的β pruning。

在一个极大节点n，设β是目前已经检查的n的兄弟节点的最小值，α是n的已经检查的子节点的最大值，当α≥β时，我们就可以停止检查n的剩余子节点。因为极小节点Min永远不会选择从n的父母移到n，因为它会首先选择n的低价值兄弟姐妹之一。



那么对于一个极小节点m一样，设α时目前已经检查的m的兄弟节点的最大值，β是m已经检查的子节点的最小值，当β≤α时，我们就可以停止检查m剩余的子节点了，因为最大节点Max永远不会选择从n的父级移至n，因为它将首先选择n的较高值的同级之一。



更一般的来讲，我们上述所说的所有父节点都可以推广到祖先节点。

下面我们用伪代码的形式更加清晰地说明α-β剪枝算法：

```
AlphaBeta(n,Player,alpha,beta) //return Utility of state
    If n is TERMINAL
        return V(n) //Return terminal states utility
    ChildList = n.Successors(Player)
    If Player == MAX
        for c in ChildList
            alpha = max(alpha, AlphaBeta(c,MIN,alpha,beta))
            If beta <= alpha
                break
        return alpha
    Else //Player == MIN
        for c in ChildList
            beta = min(beta, AlphaBeta(c,MAX,alpha,beta))                If beta
<= alpha
                break
        return beta
```

3. Codes

Question 1

```
 start = problem.getStartState()

    knownstates = []
    states = util.PriorityQueue()
    states.push((start,[]),heuristic(start,problem))
    while(not states.isEmpty()):
        thisstate,actions = states.pop()
        if(problem.isGoalState(thisstate)):
            return actions
        if(thisstate not in knownstates):
            successors = problem.getSuccessors(thisstate)
            for nextstate in successors:
                coordinate = nextstate[0]
                direction = nextstate[1]
                if(coordinate not in knownstates):
                    actiontopush = actions + [direction]
                    h = heuristic(nextstate[0],problem)
```

```
                    c = problem.getCostOfActions(actiontopush)
                    f = h + c
                    states.push((coordinate,actiontopush),f)
            knownstates.append(thisstate)
    return actions
```

Question 2

```
class FoodSearchProblem:
    """
    A search problem associated with finding the a path that collects all of the
    food (dots) in a Pacman game.

    A search state in this problem is a tuple ( pacmanPosition, foodGrid ) where
      pacmanPosition: a tuple (x,y) of integers specifying Pacman's position
      foodGrid:       a Grid (see game.py) of either True or False, specifying
remaining food
    """
    def __init__(self, startingGameState):
        self.start = (startingGameState.getPacmanPosition(),
startingGameState.getFood())
        self.walls = startingGameState.getWalls()
        self.startingGameState = startingGameState
        self._expanded = 0 # DO NOT CHANGE
        self.heuristicInfo = {} # A dictionary for the heuristic to store
information

    def getStartState(self):
        return self.start

    def isGoalState(self, state):
        return state[1].count() == 0

    def getSuccessors(self, state):
        "Returns successor states, the actions they require, and a cost of 1."
        successors = []
        self._expanded += 1 # DO NOT CHANGE
        for direction in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
Directions.WEST]:
            x,y = state[0]
            dx, dy = Actions.directionToVector(direction)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextFood = state[1].copy()
                nextFood[nextx][nexty] = False
                successors.append( ( ((nextx, nexty), nextFood), direction, 1) )
        return successors

    def getCostOfActions(self, actions):
        """Returns the cost of a particular sequence of actions.  If those
actions
        include an illegal move, return 999999"""
        x,y= self.getStartState()[0]
        cost = 0
        for action in actions:
            # figure out the next state and see whether it's legal
            dx, dy = Actions.directionToVector(action)
```

```
            x, y = int(x + dx), int(y + dy)
            if self.walls[x][y]:
                return 999999
            cost += 1
        return cost

class AStarFoodSearchAgent(SearchAgent):
    "A SearchAgent for FoodSearchProblem using A* and your foodHeuristic"
    def __init__(self):
        self.searchFunction = lambda prob: search.aStarSearch(prob,
foodHeuristic)
        self.searchType = FoodSearchProblem
```

Question 3

```
foodlist = foodGrid.asList()
    foodremain = list(foodlist)
    if(len(foodremain)==0):
        return 0
    h = 0
    farest1 = (0,0)
    farest2 = (0,0)
    max = 0
    for food1 in foodlist:
        for food2 in foodlist:
            if food1==food2:
                pass
            tempdistance = util.manhattanDistance(food1,food2)
            if(tempdistance>max):
                max = tempdistance
                farest1 = food1
                farest2 = food2
    if(food1==(0,0)and food2==(0,0)):
        return util.manhattanDistance(position,foodremain[0])
    h = util.manhattanDistance(farest1,farest2) +
min(util.manhattanDistance(position,farest1),util.manhattanDistance(position,far
est2))
    return h
```

Question 4

```
        def minmax_search(gameState, agentIndex, depth):
            """
            Return the evaluation value of the state of the agent
            """
            # if not the last ghost
            if agentIndex != gameState.getNumAgents():
                moves = gameState.getLegalActions(agentIndex)

                # if no actions can be done, the leaf node
                if len(moves) == 0:
                    return self.evaluationFunction(gameState)

                # get minmax value for successors
                next = []
                for this_move in moves:
```

```
 next.append(minmax_search(gameState.generateSuccessor(agentIndex,this_move),
agentIndex+1, depth))

                # if i'm max player, return max
                if agentIndex == 0:
                    return max(next)
                # if i'm min player, return min
                else:
                    return min(next)

            # if its the last ghost to act
            else:
                # if deepest explored, evaluate value
                if depth == self.depth:
                    return self.evaluationFunction(gameState)
                # if still not max depth, then it's pacman's turn and to add a
depth
                else:
                    return minmax_search(gameState, 0, depth+1)
        # end of minmax_search

        # select the action that have the gratest minmax value

        result = max(gameState.getLegalActions(0), key = lambda x:
minmax_search(gameState.generateSuccessor(0, x), 1, 1))
        return result
```

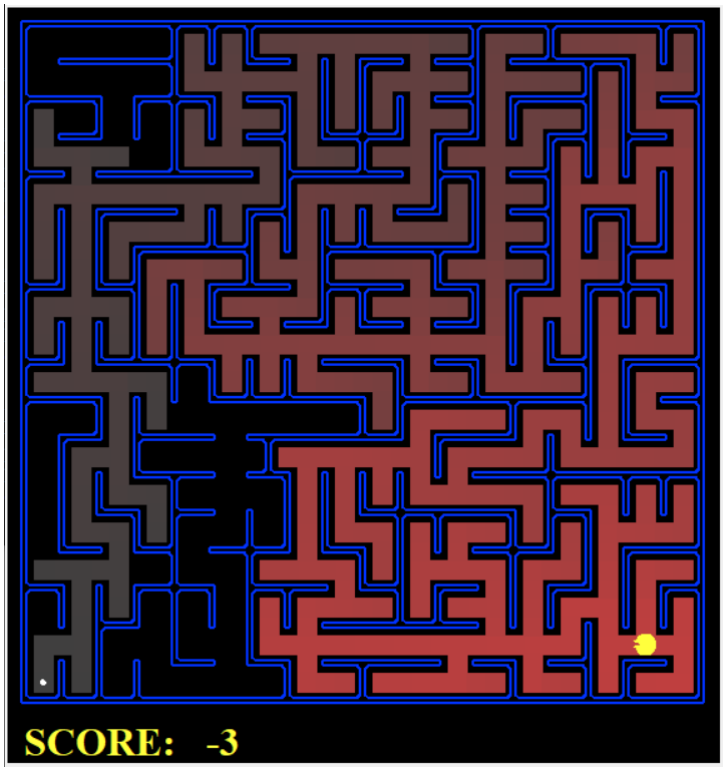
Question 5

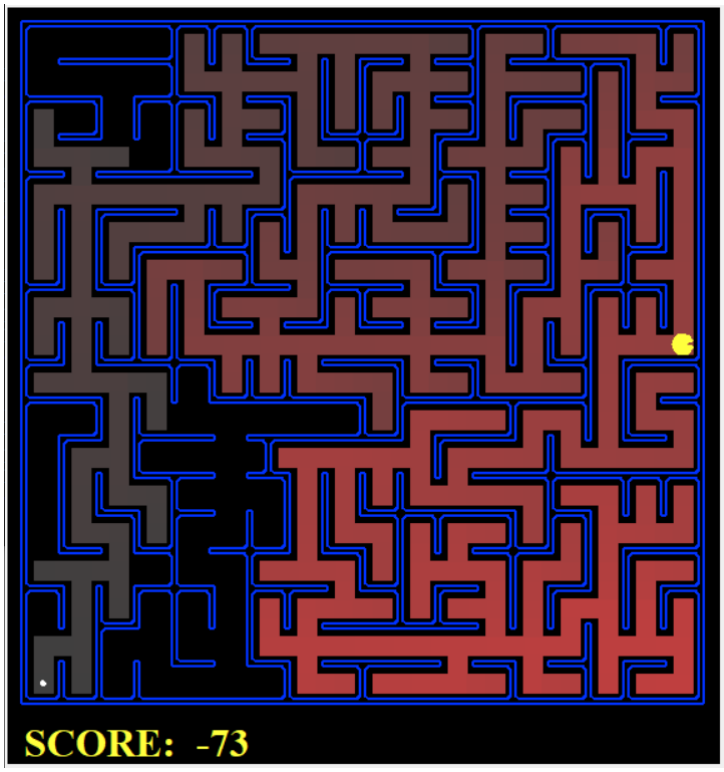
```
        def beta_value(state, index, depth, alpha, beta):
            v = 9999999
            actions = state.getLegalActions(index)
            for action in actions:
                successor = state.generateSuccessor(index, action)
                if (index + 1 == state.getNumAgents()):
                    v = min(v, alpha_value(successor, 0, depth + 1, alpha,
beta))
                else:
                    v = min(v, beta_value(successor, index + 1, depth, alpha,
beta))
                if v < alpha:
                    return v
                beta = min(beta, v)
            if (v != 9999999):
                return v
            else:
                return self.evaluationFunction(state)

        def alpha_value(state, index, depth, alpha, beta):
            if depth >= self.depth:
                return self.evaluationFunction(state)
            v = -9999999
            actions = state.getLegalActions(index)
            for action in actions:
                successor = state.generateSuccessor(index,action)
                v = max(v,beta_value(successor,index+1,depth,alpha,beta))
```

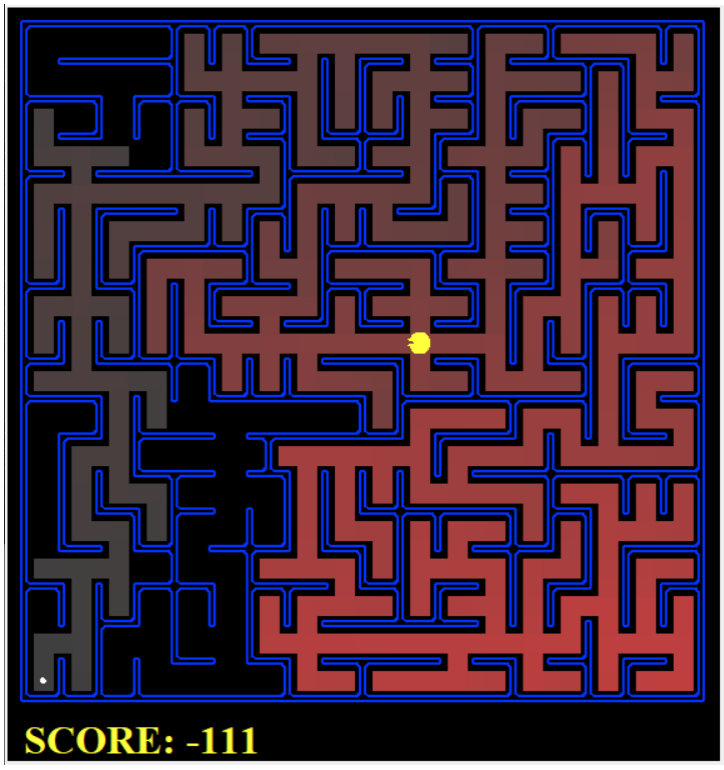
```
                if v > beta:
                    return v
                alpha = max(alpha,v)
            if v!=-9999999:
                return v
            else:
                return self.evaluationFunction(state)

        v = -9999999
        alpha = -9999999
        beta = 9999999
        bestaction = None
        actions = gameState.getLegalActions(0)
        for action in actions:
            successor = gameState.generateSuccessor(0,action)
            thisvalue = beta_value(successor,1,1,alpha,beta)
            v = max(v, thisvalue)
            if v > alpha:
                alpha = v
                bestaction = action
            #elif v==alpha and bestaction == 'Stop':
            #   bestaction = action
        return bestaction
```

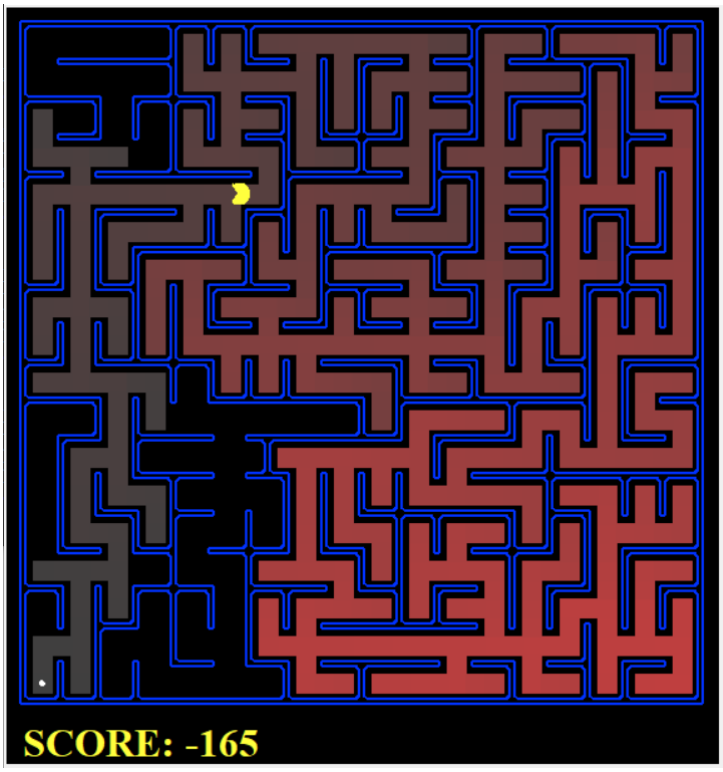
4.结果展示

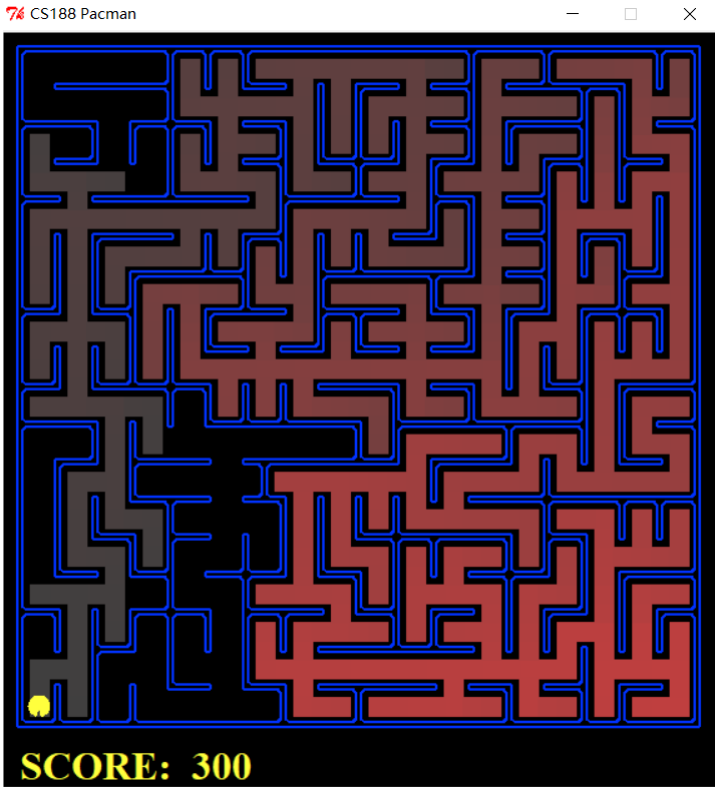
Question1




SCORE: -73

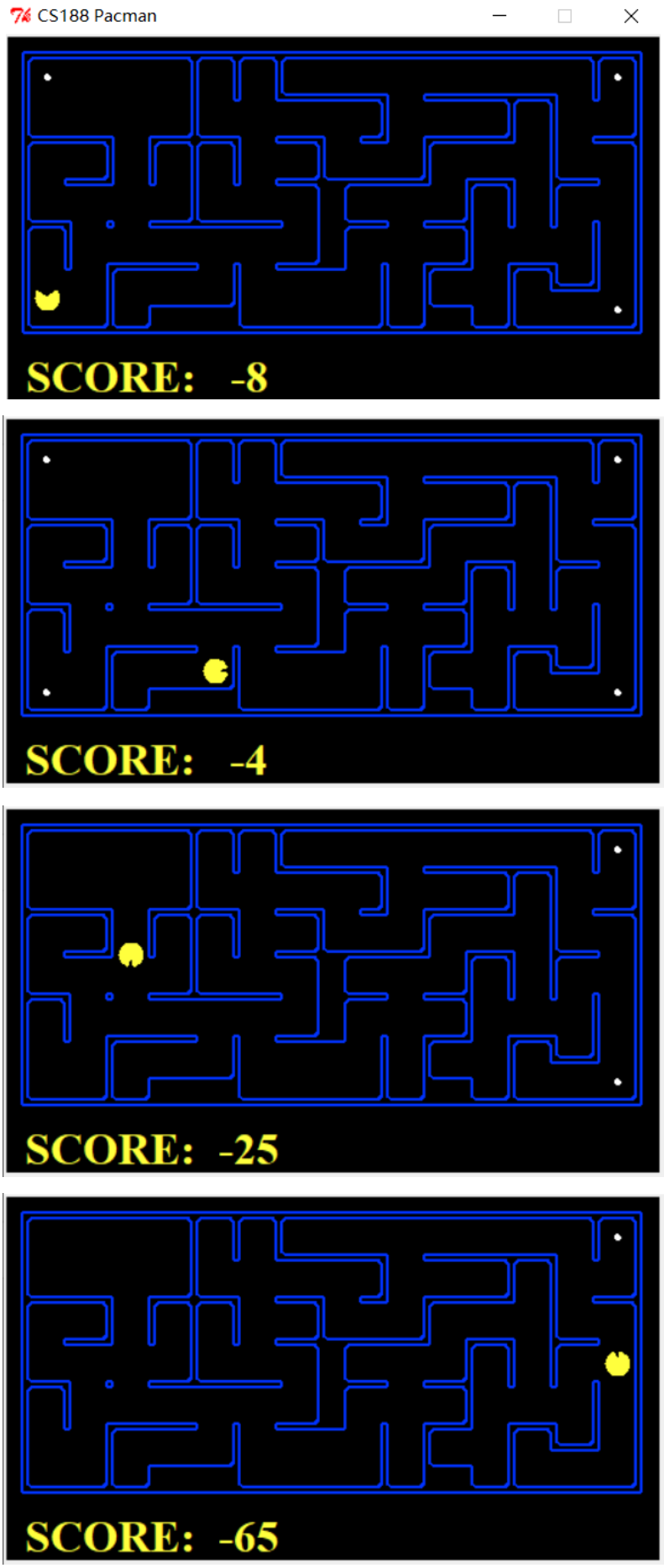

SCORE: -111






```
E:\AIEXPYTHON\P01\InforAboutPro\searchOfXM>python2 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manha
ttanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

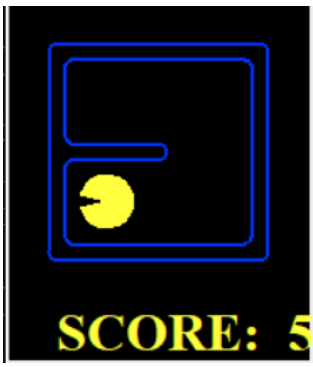
Question2


CS188 Pacman
SCORE: -8
SCORE: -4
SCORE: -25
SCORE: -65


```
E:\AIEXPYTHON\P01\InforAboutPro\searchOfXM>python2 pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 316
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

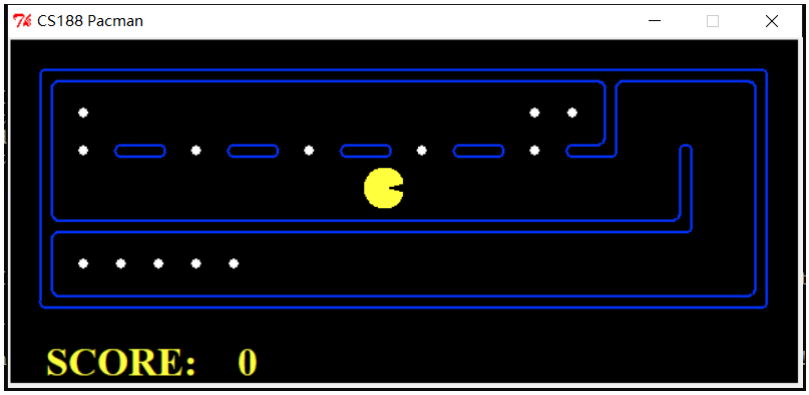
Question3

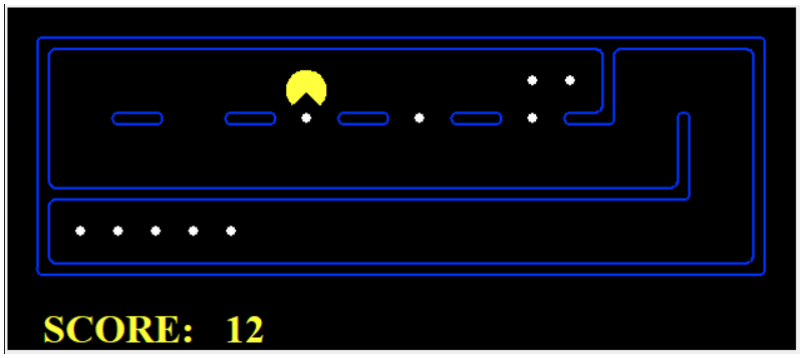
testSearch

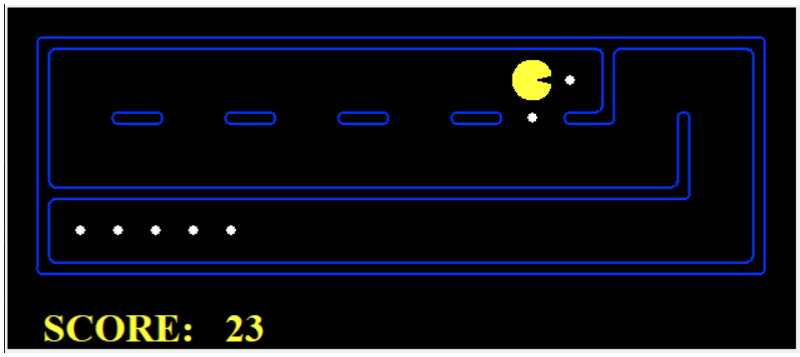


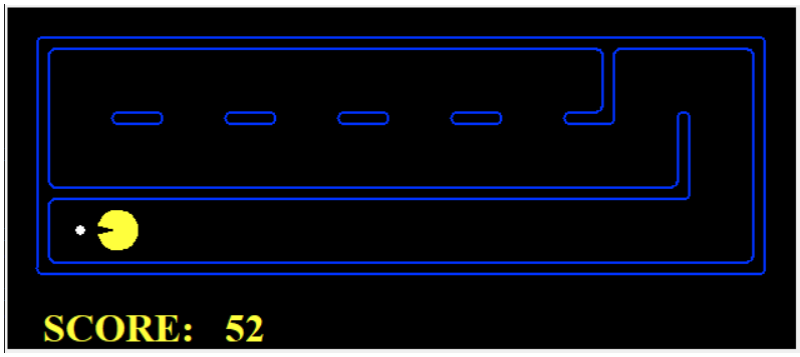

```
E:\AIEXPYTHON\P01\InforAboutPro\searchOfXM>python2 pacman.py -l testSearch -p AStarFoodSearchAgent
Path found with total cost of 7 in 0.0 seconds
Search nodes expanded: 13
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513.0
Win Rate:      1/1 (1.00)
Record:        Win
```

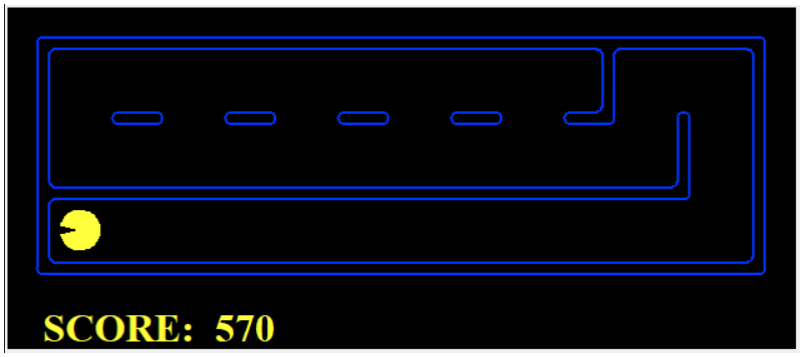
trickySearch





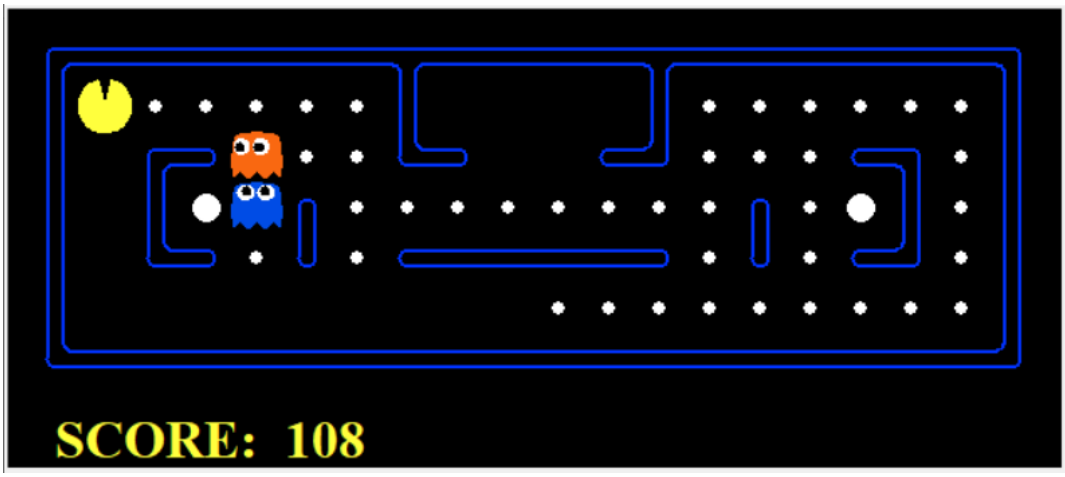


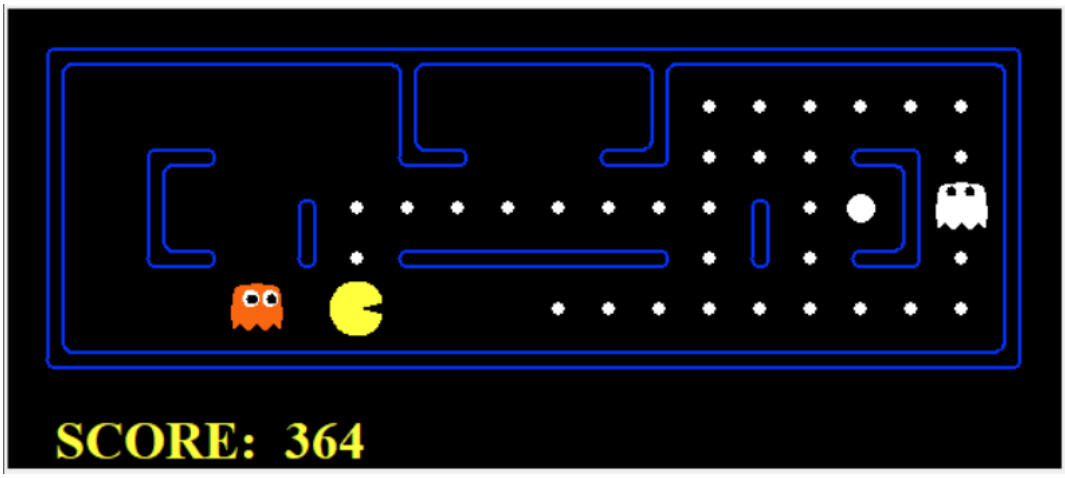


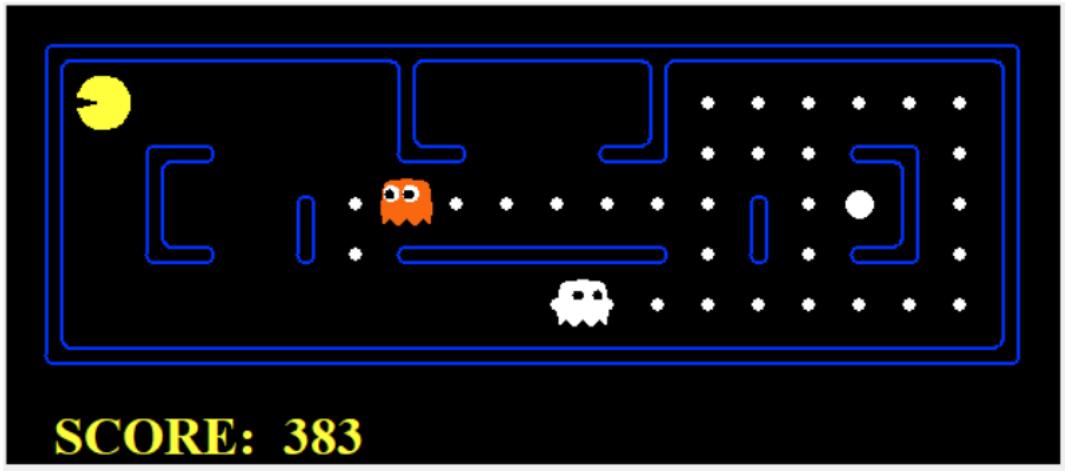


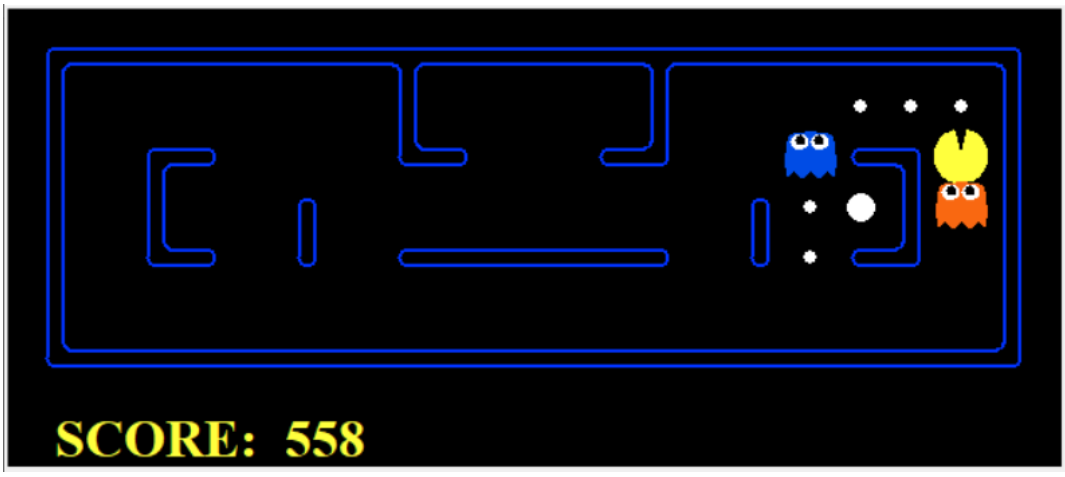

```
E:\AIEXPYTHON\P01\InforAboutPro\searchOfXM>python2 pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 8.2 seconds
Search nodes expanded: 7853
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

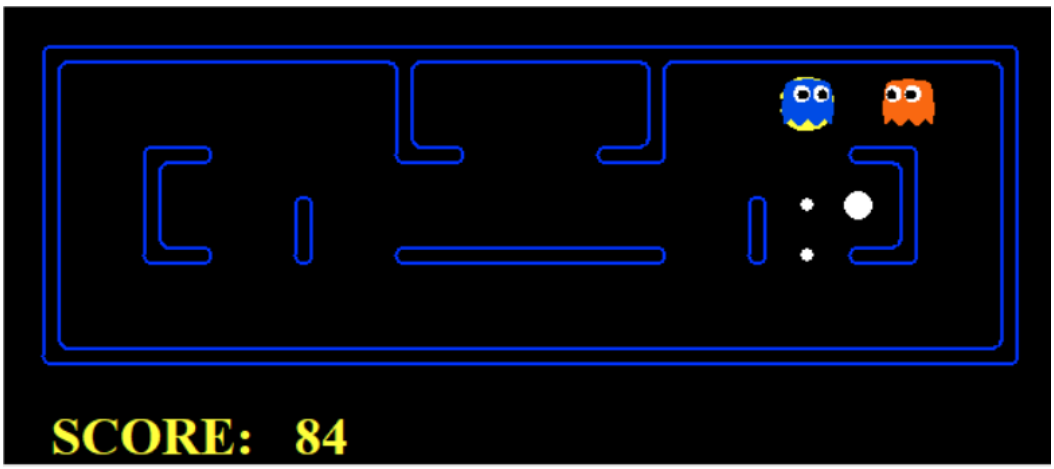
Question4





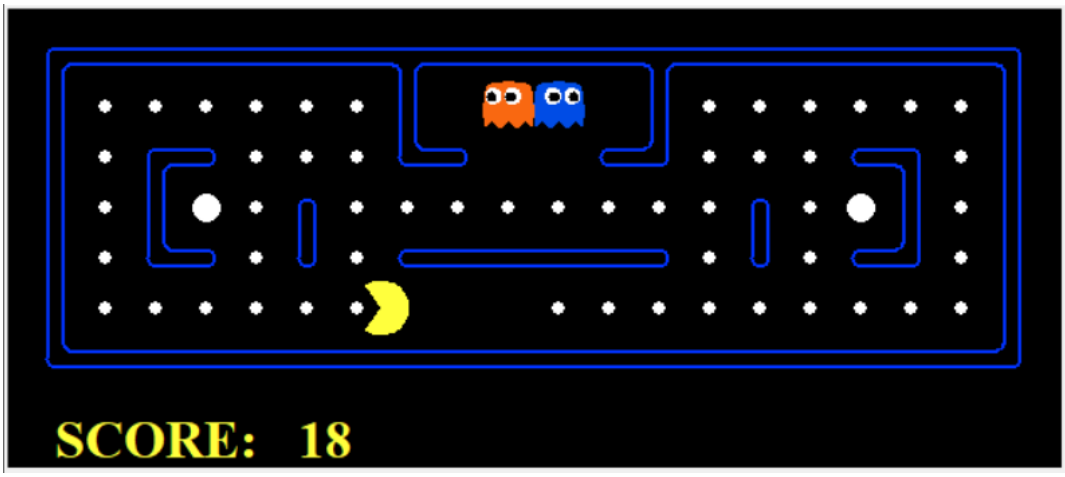


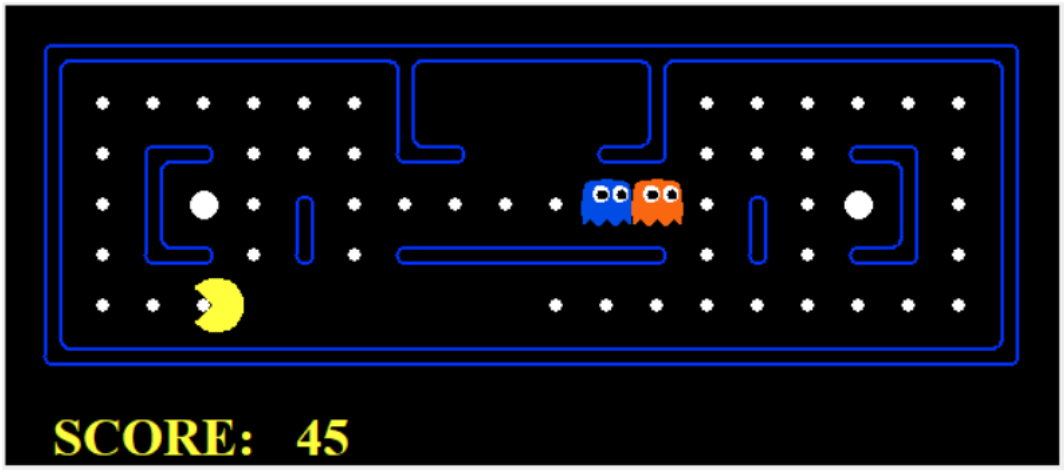


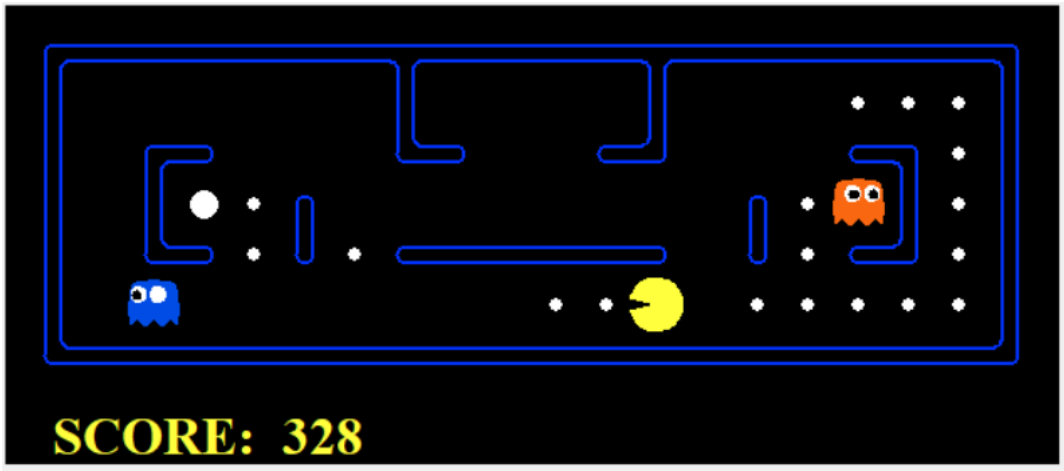

SCORE: 84

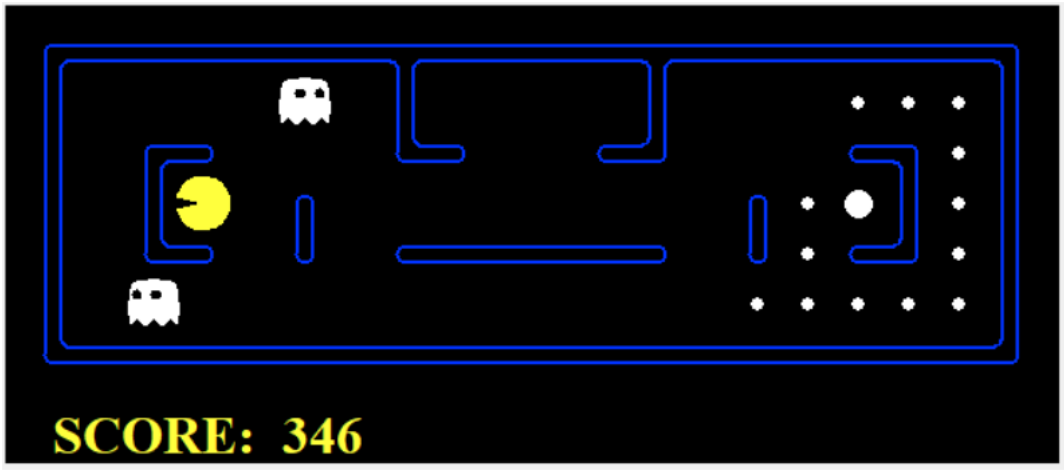
```
E:\AIEXPYTHON\P01\InforAboutPro\mutiagentOfXM>python2 autograder.py -q q2
Starting on 9-29 at 19:21:28

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-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
*** 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 28 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 19:21:56

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.
```

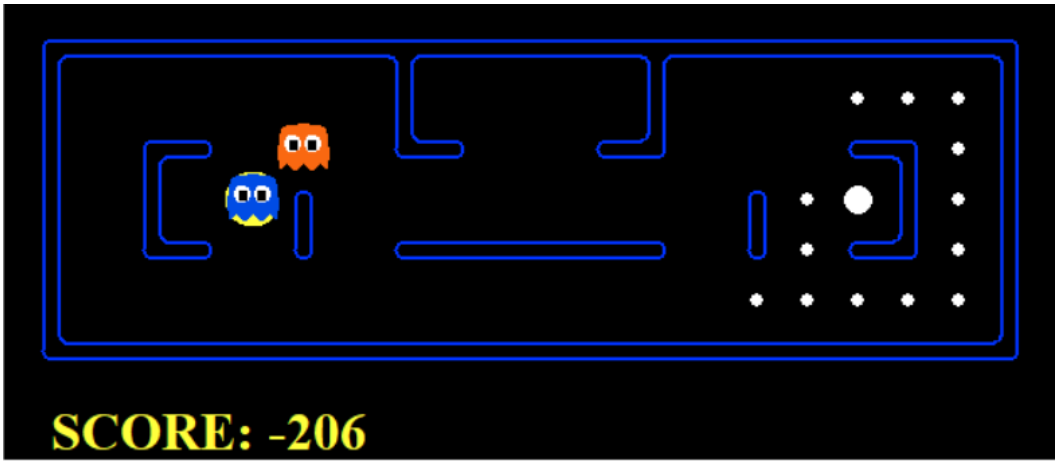
Question5


SCORE: 18


SCORE: 45


SCORE: 328


SCORE: 346




```
E:\AIEXPYTHON\P01\InforAboutPro\mutiagentOfXM>python2 pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
Pacman died! Score: -206
Average Score: -206.0
Scores:        -206.0
Win Rate:      0/1 (0.00)
Record:        Loss
```

5.结果分析

1.Search in Pacman

Question1

第一个问题的启发式函数我们使用的是简单的曼哈顿距离，也就是当前位置和目标位置横纵坐标只差的绝对值之和作为启发式函数。这是一个非常普遍常见的启发式函数，下面我们来叙述这个启发式函数的有效性和一致性，因为一致性可以保证有效性，所以下面我们只证明一致性。

$$
\begin{gathered}
\text{设以曼哈顿距离作为标准启发式函数为} h(n) \\
\text{欲证明一致性只需证明对顺序访问的节点} n1, n2 \text{满足} h(n1) \leq h(n2) + c \\
\text{因为每次移动一步，曼哈顿距离最多减少} 1\text{，而我们的} c \equiv 1\text{，所以上式显然成立，一致性得证。}
\end{gathered}
$$

因为以曼哈顿距离作为启发式函数满足一致性，所以可以保证最优性和完备性，设计合理。

Question2

第二个问题我们设计的启发式函数原理如下：首先计算距离当前位置最近的一个角的曼哈顿距离，不妨就设这个角为`corner1`，设起始位置到`corner1`的曼哈顿距离为`man1`。之后计算距离`corner1`曼哈顿距离最小的一个角，不妨就设为`corner2`，`corner1`到`corner2`的曼哈顿距离为`man2`。再计算距离`corner2`曼哈顿距离最小的一个角，不妨就设为`corner3`，`corner2`到`corner3`的曼哈顿距离为`man3`。最后计算`corner3`到最后一个角的曼哈顿距离，设最后这个角为`corner4`，曼哈顿距离为`man4`,我们最终的启发式函数的估值即为：`man1+man2+man3+man4`。同样我们对这个启发式函数证明它的一致性。

$$
\begin{gathered}
\text{设以上述估值作为标准启发式函数为} h(n) \\
\text{欲证明一致性只需证明对顺序访问的节点} n1, n2 \text{满足} h(n1) \leq h(n2) + c \\
\text{因为每次移动一个位置，只可能对四个角上的食物曼哈顿距离之和最多减少一} \\
\text{而我们的} c \equiv 1\text{，所以上式显然成立，一致性得证。}
\end{gathered}
$$

Question3

在第三个问题中我们设计的启发式函数原理如下：如果地图中只有一个豆子，那么我们的估值就是当前位置到这个豆子的曼哈顿距离。否则首先计算曼哈顿距离最远的两个食物的曼哈顿距离，记为`man1`，之后计算我们到这两个食物之中更近的一个食物的曼哈顿距离，记为`man2`，我们的启发式函数估值即为`man1+man2`。根据启发式函数的设计准则，我们可以将问题进行简化然后进行估值，则对于这个问题我们可以简化成在当前的地图中只有相距最远的两个豆子，那么我们最好的选择一定是先去吃较近的一个，再去吃较远的一个。下面我们同样证明这个启发式函数的一致性

如果地图中只有一个豆子，那么估值函数即为曼哈顿距离，

一致性我们已经在第一问中证明，在此不再赘述

设$n1$状态下最远的两个豆子的曼哈顿距离为$man1(n1), n1$距离较近的豆子的曼哈顿距离为$man2(n1)$

那么同理$n2$状态下最远的两个豆子的曼哈顿距离为$man1(n2), n2$距离较近的豆子的曼哈顿距离为$man2(n2)$

设估值函数为$h(n) = man1(n) + man2(n)$

我们可以知道对于同一个地图，只要当前状态还有两个豆子，那么就有对任意的状态$n1, n2, man1(n1) = man2(n2)$

而对于$man2,$因为每次移动一步，曼哈顿距离最多减少1，而我们的$c \equiv 1$

所以可得

$$h(n1) = man1(n1) + man2(n1) = man1(n2) + man2(n1) \leq man1(n2) + man(n2) + c = h(n2) + c$$

一致性得证

2.Multi-Agent Pacman

- 我们规定**b**代表一个状态的最大后继数，也就是分支银子
- 规定**d**为博弈树的最大深度
- **minmax算法**是在假设所有玩家都做出最理性的选择的前提下进行的，对于max节点来说会选择下方min节点中最大效益值的动作，而min节点会做出下方状态中效益值最小的动作。我们用的是DFS的方式实现：
  - **空间复杂度**：由于需要建造出整颗博弈树，空间复杂度为DFS的线性空间复杂度，也就是栈的最大深度，为

    $$O(bd)$$

  - **时间复杂度**：需要搜索完整棵博弈树才可以找到最优的策略，否则可能没有考虑到部分的min或max的效益值，所以时间复杂度是整棵博弈树的节点数量级

    $$O(b^d)$$

- **α-β剪枝算法**能够在上面的minmax的基础之上，剪除不可能为最优效益的路径，不再扩展，算法在搜索时维持当前MAX节点最优值alpha和当前MIN节点最优值beta。当节点的效用值比MAX节点的alpha或MIN节点的beta更差时，剪枝掉该节点的分支，节约搜索时间。由此大大提升我们的搜索性能：
  - **空间复杂度**：由于仍然与DFS的思路一脉相承，空间复杂度的度量是栈的最大深度，是线性的空间复杂度

    $$O(bd)$$

  - **时间复杂度**：最优情况下，也就是最优解被优先搜索的时候，α-β剪枝算法比minmax的搜索节点少一半，因此时间复杂度为

    $$O(b^{d/2})$$

6.Experimental experience

1. 本次第一个实验，做的是吃豆人项目，整个感觉下来是很奇妙的体验。首先是疫情过去很久没有这样合作写过项目了，和小伙伴一起组队有时没有提前沟通好一些接口，导致在A*和角落问题的时候，对于传入参数的定义不大一致，导致后来调试了很久改了很久的问题，最后还是推倒重写了。所以以后得磨刀不误砍柴工，在商量好思路之后对于需要合作的参数接口要一起制定，会节省后期的排错时间
2. 对于Question3我们尝试了三个不同的启发式函数：
   - 计算曼哈顿距离最远的两个食物的曼哈顿距离，记为`man1`，之后计算我们到这两个食物之中更近的一个食物的曼哈顿距离，记为`man2`，我们的启发式函数估值即为`man1+man2`
   - 计算最远的豆子到当前位置的曼哈顿距离`man1`，以及相对于当前位置，和最远的位置处于异侧的的豆子的数目`count`，启发式函数估值为`man1+count`

- 计算当前状态到距离最近的豆子的曼哈顿距离`man1`，以及这个状态到除了距离最近的豆子，和其他豆子的曼哈顿距离的平均值`avg`，最终的估值函数为`man1 + avg`

通过测试这三个估值函数，我们发现性能逐渐递减，也就是说第一个函数的性能最好，其次是第二个，第三个最差。第三个估值函数搜索的节点数甚至达到了13000个以上，但是第一个估值函数只需要搜索7800个节点左右，所以我们可以得知，设计一个好的启发式函数对算法性能的提升是重大的。

3. 对于Question5，我们在实现的过程中调试了很久，算法上看没有问题但是结果计算的估值就是不正确，最后是因为传递参数的原因。python2传递的参数是这个参数的指针而不是副本，所以一定要更改参数的传递才能正确运行
4. 我们觉得这个Pacman项目的确是很完备很好的项目体验，对于文档的注释以及说明很完善，自己要做的任务以及接口只要用心看都很清晰明白，能够很好锻炼能力，我们还是很喜欢这个项目的
5. 通过对不同算法的学习体会，我们在一步步完善项目的过程中也将理论课以及之前小实验的所习汇总成一个大的体系，对比中了解不同，反思后才得优化，算法之间也是由于一代代对性能的提升需求才一步步进化，比如剪枝优化了搜索节点有些并不必要的劣势。我们也应当保持一双审视的眼睛，看到学习生活中那些可以更加提升的地方，并尝试有没有改进的思路。