同济大学计算机系

人工智能课程设计实验报告

学　　号 2152809

姓　　名 曾崇然

专　　业 计算机科学与技术

授课老师 武妍老师

一. 问题概述

1. 问题的直观描述

总体：

这个项目中，我将设计经典版 Pacman 的智能体，包括幽灵。在此过程中，我将实现极小化和期望最大化搜索，并尝试设计评估函数。

具体：

问题 1：设计一个评估函数，对一个给定状态可能的行动进行评分，越好的行动应当赋予它越高的分数值

问题 2：设计一个函数，能够从游戏当前的状态开始扩展指定层数的最大最小搜索树，用给定的 evaluationFunction 函数评估叶子结点的值，并根据是 max 层还是 min 层确定叶子结点之外的结点的值，最后返回根节点的值并返回使得根节点取得最值的对应行动

问题 3：在问题 2 的基础上应用 α - β 剪枝算法，将不用扩展的结点剪除，提高搜索的效率

问题 4：Minimax 和 alpha-beta 算法是假设对手始终会选择最优的对策，但是实际情况往往不是这样的，因此上述两种算法有时候会导致游戏失败和分数更低，问题 4 要求设计一个搜索算法能够更好的应对这样一种情况：对手不是总选择最优的行动，而是在可能的行动中等可能随机选择

问题 5：设计一个评估函数对某一个状态进行评估而不是对某一个状态的的行为进行评估，并使其满足给定的要求

2. 已有代码的阅读和理解

① 由于在问题要求完成的类继承了 game.py 中的 Agent 类，所以需要对 Agent 类进行理解：Agent 类是一个抽象类，给出了构造函数，表明该 Agent 是几号智能体；并定义了一个函数接口 getAction，这个函数将接受一个状态并返回一个可能的行动(好的行动)，具体实现由继承 Agent 的不同子类实现

② 由于函数和类的完成是在给定游戏状态 GameState 的基础上进行的，所以在完成的过程中需要使用 GameState 中的属性和方法，有必要对 pacman.py 中的 GameState 类进行理解：GameState 中给出了返回该状态可能行为的的方法 getLegalActions，返回对应方法下一个状态的方法 generateSuccessor，以及返回吃豆人、鬼、食物、墙、当前分数等一系列用于帮助类和函数完成的方法。

3. 解决问题的思路和想法

问题 1：某个行为的好坏取决于该行为导致的下一个状态，这应当和豆子的剩余数量，距离豆子的距离，吃豆人和鬼的距离等因素决定，豆子剩余数量越少，距离豆子距离越近，距离鬼的距离越远，评分就应当越高

问题 2：从当前状态开始，递归地扩展最大最小搜索树，使用评估函数评估叶子结点，再根据是 min 结点或者 max 结点从底层向顶层确定其他结点的值，最后返回当前状态的最值以及取得该最值的对应行动

问题 3：在问题 2 的基础上除叶子结点的每个结点都对应 alpha 和 beta 值，如果 min 结点的无法更新父辈结点的 alpha 值或者 max 结点无法更新父辈结点的 beta 值，则无需再访问该节点的其他子节点(剪掉)

问题 4：由于鬼是随机选择可能的行动，因此 min 结点的评估值不该是其最小的子节点的值，由于是均匀随机选择，所以我选择将其所有子节点值的平均值作为该 min 结点的评估值

问题 5：该问题和问题 1 由相似之处，使评估函数更加优秀，我添加了当前的分数，周

围墙的数量，随机数，并将最短的豆子的距离取代为：最近的豆子距离 min，距离该豆子最远的豆子两个豆子之间的距离 max，这两个值的和 min+max，用以上量来对某个状态进行评估

二. 算法设计

问题 1：

算法功能：给定状态和行动，对该行动进行评分，该行动越有利于吃豆人，分数越高

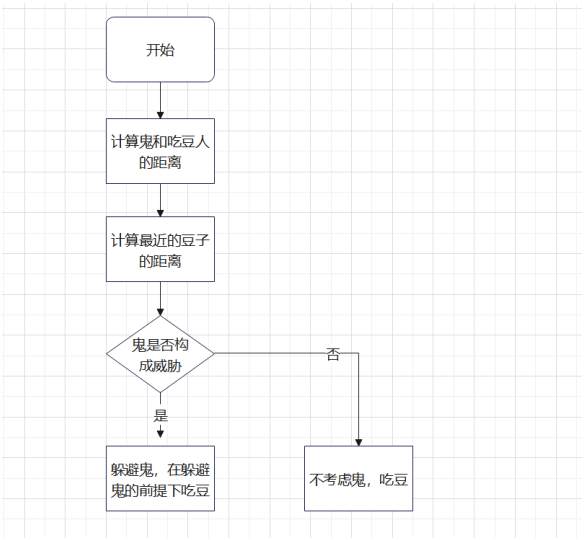
设计思路：①为了使吃豆人能够躲避鬼，应当将鬼距离吃豆人的距离考虑在内

②为了使吃豆人能够吃到豆子，应当将最近的豆子的距离和剩余豆子的数量考虑在内

③当鬼的距离不足以构成威胁时，主要目标是吃掉豆子，当鬼的距离已经构成威胁时，应当以躲避鬼为主要目标

④在躲避鬼时，可能有多种方案，应当从中选择利于吃到豆子的

流程图：



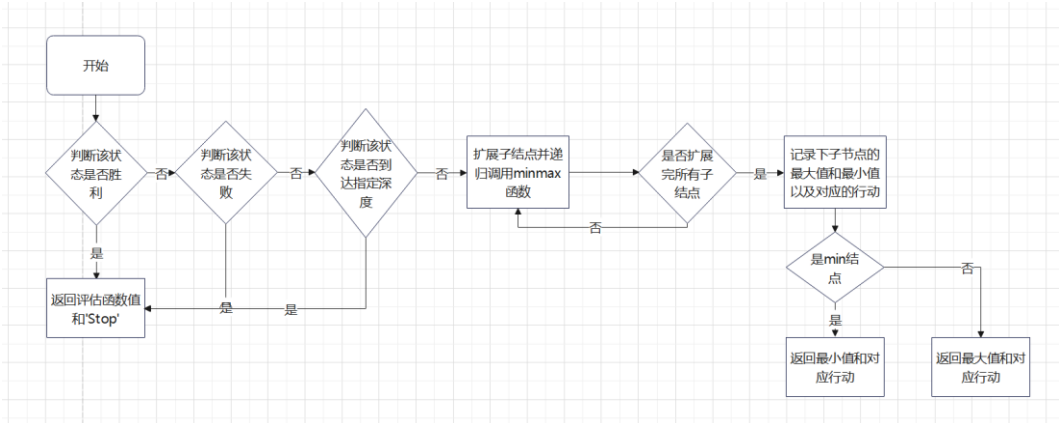
问题 2：

算法功能：如果一个结点是 min/max 结点，则根据子节点中的最小值/最大值设置该结点的值，并返回该值和取得该值对应的行动

设计思路：① 由于子节点的值也需要调用该函数，所以是一个递归的结构。

② 根据当前树的层数和要求扩展的深度来判断是 min 结点还是 max 结点

流程图：

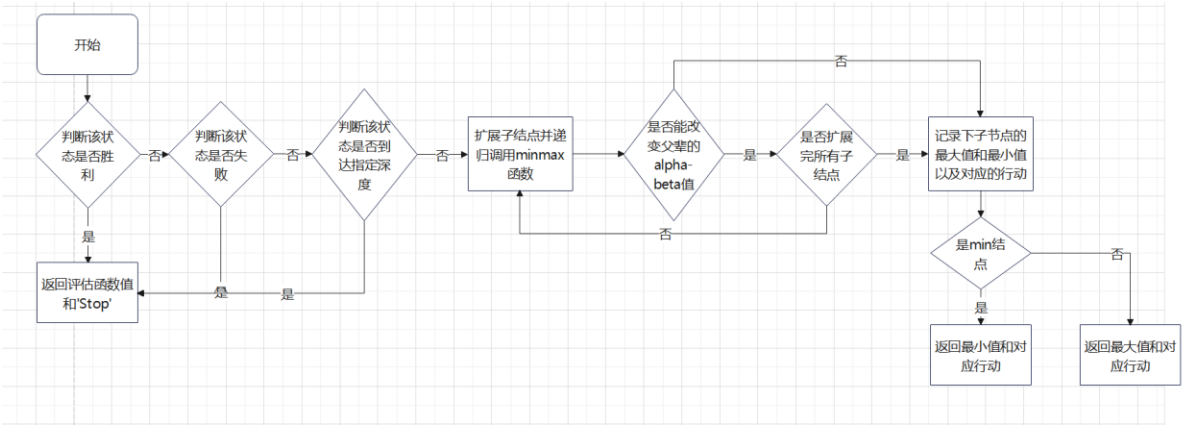


问题 3：

算法功能：在实现算法 2 功能的基础上，根据每个结点的 alpha 值和 beta 值来省去不必要的结点扩展（alpha=beta 剪枝）

设计思路：在算法 2 的基础上将父辈 max 结点对应的 alpha 值列表和父辈 min 结点对应的 beta 值列表作为参数传入函数，如果当前 min 结点已经不能再改变父辈的 alpha 值或者当前 max 结点已经不能再改变父辈的 beta 值，则不再扩展该结点的剩余结点，实现剪枝

流程图：

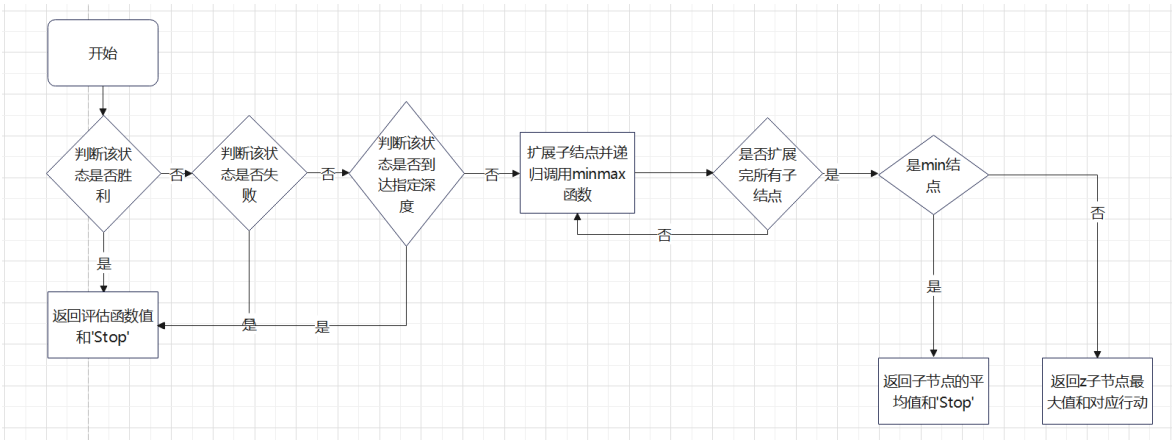


问题 4：

算法功能：吃豆人中鬼的行动是均匀随机的，并不是总是选择最有利的行动，要求设计的算法能够根据这种特点返回吃豆人在某个状态的下一个最优行动

设计思路：由于鬼的行动是随机的，因此鬼并不一定向扩展出的最不利于吃豆人的状态行动，而是所有行动都可能采取，并且等可能，所以用 min 结点的子节点的平均值作为 min 结点的值

流程图



问题 5：

算法功能：评估一个给定的游戏状态，返回评估的分数

设计思路：为了能够更好的评估一个游戏状态，需要将以下内容考虑在内：

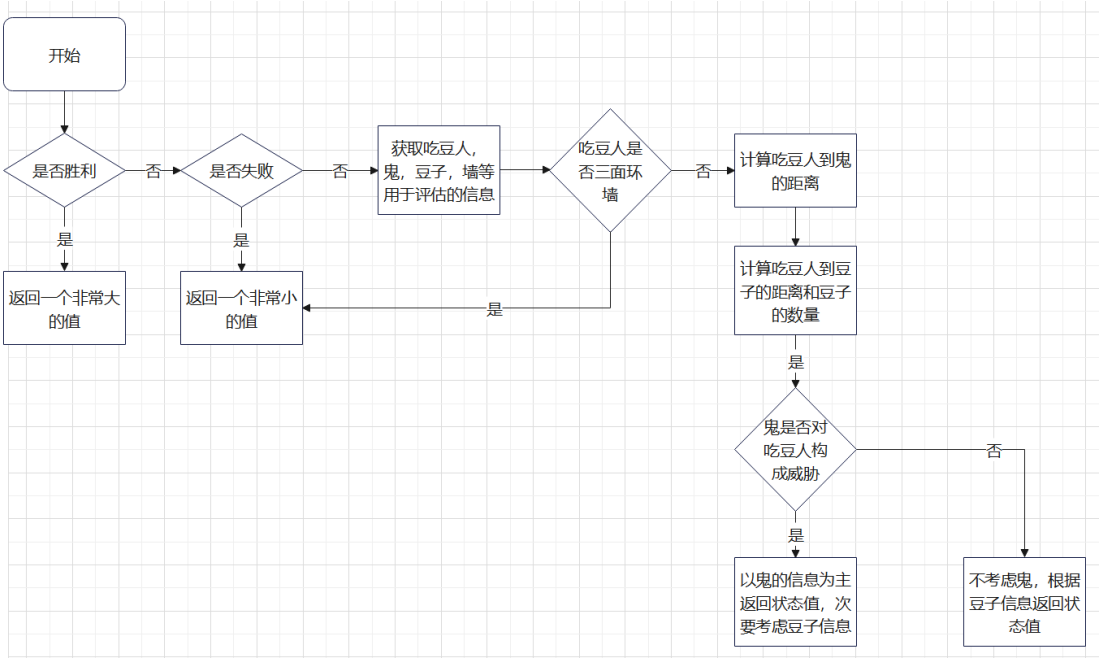
① 鬼和吃豆人的距离，因为只要一个鬼碰到吃豆人就代表游戏失败，因此用和吃豆人距离最近的鬼的距离来代表和鬼的距离，该距离越近，游戏状态的评分应当越低

② 豆的数量和与吃豆人的距离，因为吃豆人的目标是将所有豆子都吃完，所以豆子的数量越少，吃豆人距离豆子的距离越近，状态评分应当越高。为了相

对于问题 1 更好的评估和豆子的距离，我将最近的豆子距离 min，距离该豆子最远的豆子两个豆子之间的距离 max，这两个值的和 min+max，来代表和吃豆人和豆子之间的距离

③ 墙的位置，如果一个状态是吃豆人的三面有墙，则它不得不返回之前的位置，如果这个位置没有豆子，那么这就是无效行动，我将这种状态的评分给的很低以避免吃豆人陷入反复的来回运动

流程图：



三. 算法实现

问题 1：

代码：

```
    GhostLocation=newGhostStates[0].getPosition()
    x=abs(newPos[0]-GhostLocation[0])
    y=abs(newPos[1]-GhostLocation[1])
    dGhost=x+y#吃豆人和鬼的距离

    min=10000
    FoodLocation=newFood.asList()
    for i in FoodLocation:
        x = abs(newPos[0] - i[0])
        y = abs(newPos[1] - i[1])
        temp=x+y
        if min>temp:
            min=temp#最近的豆子

    num = newFood.count()

    if dGhost<=2:
        Evaluation=dGhost*100#如果鬼的距离过近，则以鬼为主要评价因素
        if num == 0:
            Evaluation = Evaluation+10
        else:
            Evaluation = Evaluation-num * 1 + 1.0 / min#次要以豆的距离为评价因素
        Evaluation=Evaluation-10000
    else:
        if num==0:
            Evaluation=10000
        else:
            Evaluation=-num*100+100.0/min#如果鬼的距离不近，以豆子信息为评价因素
    return Evaluation
```

实现细节：①用最近的豆子的距离模拟豆子的距离

②当鬼和吃豆人的距离小于等于 2 时，主要通过鬼的距离进行评估，次要通过豆子的数量和位置评估；当鬼和吃豆人的距离大于 2 时，不考虑鬼的影响，只通过豆子的信息进行评估

③保证大于 2 的评估值总是大于小于等于 2 的评估值

模块输入：通过 GameState 类的方法获取游戏信息的相关状态

数据结构定义：无

问题 2：

代码：

```
def getAction(self, gameState: GameState):
    """
    Returns the minimax action from the current gameState using self.depth
    and self.evaluationFunction.

    Here are some method calls that might be useful when implementing minimax.

    gameState.getLegalActions(agentIndex):
    Returns a list of legal actions for an agent
    agentIndex=0 means Pacman, ghosts are >= 1

    gameState.generateSuccessor(agentIndex, action):
    Returns the successor game state after an agent takes an action

    gameState.getNumAgents():
    Returns the total number of agents in the game

    gameState.isWin():
    Returns whether or not the game state is a winning state

    gameState.isLose():
    Returns whether or not the game state is a losing state
    """
    "*** YOUR CODE HERE ***"
    numAgents=gameState.getNumAgents()
    value=self.minmaxSelect(gameState,numAgents,0)
    return value[1]
```

```
def minmaxSelect(self,gameState: GameState,numAgents,layer):
    actionChoice='Stop'
    if gameState.isLose():
        return [self.evaluationFunction(gameState),'Stop']
    if gameState.isWin():
        return [self.evaluationFunction(gameState), 'Stop']#如果胜利或者失败，则不必递归下去
    if layer==numAgents*self.depth:
        return [self.evaluationFunction(gameState),actionChoice]#递归到最后一层时结束
    else:
        max=-100000
        maxAction='Stop'
        min=100000
        minAction='Stop'
        legalActions=gameState.getLegalActions(layer%numAgents)
        for action in legalActions:
            legalState=gameState.generateSuccessor(layer%numAgents, action)
            value=self.minmaxSelect(legalState,numAgents,layer+1)
            if max<value[0]:
                max=value[0]
                maxAction=action
            if min>value[0]:
                min=value[0]
                minAction=action#求出子状态的最大最小值和对应的行动
        if layer%numAgents==0:
            return [max,maxAction]
        else:
            return [min,minAction]#根据结点的类型返回对应值
```

实现细节：①扩展的树的深度等于要求扩展的深度与智能体数量的乘积，所以递归结束的条件就是树的深度等于要求深度与智能体数量的乘积

②通过树的层数来模智能体的数量来判断是哪个智能体对应的状态，依次判断是 min 结点还是 max 结点

核心函数：minmaxSelect(self,gameState: GameState,numAgents,layer)

因为需要递归调用，同时函数的参数需要智能体数量和当前层数，所以单独在类里定义了一个函数，作用是返回某个结点的 minmax 值和该值对应的行动

模块输入：GameStata 类里的各种获取当前状态信息的方法

数据结构定义：逻辑结构是树，递归函数按照树的逻辑结构不断扩展子状态，直到到达指定深度

问题 3：

代码：

```
  def getAction(self, gameState: GameState):
      """
      Returns the minimax action using self.depth and self.evaluationFunction
      """
      "*** YOUR CODE HERE ***"
      numAgents=gameState.getNumAgents()
      value=self.alphaBetaSelect(gameState,0,numAgents,[-100000],[100000])
      return value[1]
```

```
def alphaBetaSelect(self,gameState: GameState,layer,numAgents,pAlpha:list,pBeta:list):
    """
    :param pAlpha: 父辈max结点的alpha值
    :param pBeta: 父辈min结点的beta值
    """
    if gameState.isLose():
        return [self.evaluationFunction(gameState),'Stop']
    if gameState.isWin():
        return [self.evaluationFunction(gameState), 'Stop']
    if layer==numAgents*self.depth:
        return [self.evaluationFunction(gameState), 'Stop']#同问题2
    alpha=-100000
    alphaAction='Stop'
    beta=100000
    betaAction='Stop'
    legalActions = gameState.getLegalActions(layer % numAgents)
    for action in legalActions:
        if layer % numAgents == 0:
            pAlpha.append(alpha)
        else:
            pBeta.append(beta)
        legalState = gameState.generateSuccessor(layer % numAgents, action)
        tem = self.alphaBetaSelect(legalState, layer + 1, numAgents, pAlpha, pBeta)#将该状态的alpha值或者beta值加入列表进行递归
        if layer%numAgents==0:
            if tem[0]>alpha:
                alpha=tem[0]
                alphaAction=action
        else:
            if tem[0]<beta:
                beta=tem[0]
                betaAction=action
```

```
        if layer % numAgents == 0:
            pAlpha.pop()
        else:
            pBeta.pop()
        flag=0
        if layer%numAgents!=0:
            for x in pAlpha:
                if beta<x:
                    flag=1
                    break
        else:
            for x in pBeta:
                if alpha>x:
                    flag=1
                    break
        if flag==1:
            break#如果不能更新父辈的alpha值或者beta值，则不必再扩展子节点
    if layer%numAgents==0:
        return [alpha,alphaAction]
    else:
        return [beta,betaAction]
```

实现细节：①将父辈结点的 alpha\beta 值存放在列表中作为参数传入函数，当子节点无法更新父辈结点的值时，就不再扩展该结点的子节点

②更新结点的时机是子节点返回值时，然后更新列表里的值，用于剪枝

核心函数：alphaBetaSelect(self,gameState: GameState,layer,numAgents,pAlpha:list,pBeta:list)，pAlpha 和 pBeta 是父辈结点的 alpha 值和 beta 值的列表，用于判断某个状态是否还需要扩展子结点

模块输入：GameState 类中定义的各种用于获取游戏状态信息的方法

数据结构定义：树，同第二题

问题 4：

代码：

```python
def getAction(self, gameState: GameState):
    """
    Returns the expectimax action using self.depth and self.evaluationFunction

    All ghosts should be modeled as choosing uniformly at random from their
    legal moves.
    """
    "*** YOUR CODE HERE ***"
    numAgents = gameState.getNumAgents()
    value = self.ramdomSelect(gameState, numAgents, 0)
    return value[1]
```

```python
def ramdomSelect(self,gameState: GameState,numAgents,layer):
    actionChoice = 'Stop'
    if gameState.isLose():
        return [self.evaluationFunction(gameState), 'Stop']
    if gameState.isWin():
        return [self.evaluationFunction(gameState), 'Stop']
    if layer == numAgents * self.depth:
        return [self.evaluationFunction(gameState), actionChoice]
    else:
        max = -100000
        maxAction = 'Stop'
        min = 0
        minAction = 'Stop'
        legalActions = gameState.getLegalActions(layer % numAgents)
        i=0
        for action in legalActions:
            legalState = gameState.generateSuccessor(layer % numAgents, action)
            value = self.ramdomSelect(legalState, numAgents, layer + 1)
            if max < value[0]:
                max = value[0]
                maxAction = action
            min=min+value[0]
            i=i+1
        min=min/i#如果是min结点则该结点的值为子节点值的平均值
        if layer % numAgents == 0:
            return [max, maxAction]
        else:
            return [min, minAction]
```

实现细节：因为鬼是均匀随机的选择可能的行动，因此在评估原来 minmax 树中 min 结点的值时不能够再选取最小值，而是应该取所有子节点的平均值（等可能）

核心函数：ramdomSelect(self,gameState: GameState,numAgents,layer)，参数的含义与第二层相同，这是因为该函数只是将 min 结点的评估值改为了所有子结点的平均值

模块输入：GameState 类中定义的各种获取游戏状态信息的方法

数据结构定义：树，由于只修改了评估 min 结点值的方式，所以逻辑结构也是树

问题 5：

代码：

```python
def betterEvaluationFunction(currentGameState: GameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).

    DESCRIPTION: <write something here so we know what you did>
    """
    "*** YOUR CODE HERE ***"
    if currentGameState.isWin():
        return 10000+random.random()#使吃掉最后一个后吃豆人知道去哪个地方，防止停在最后一个豆子前
    if currentGameState.isLose():
        return -10000
    walls=currentGameState.getWalls()
    pacmanPosition = currentGameState.getPacmanPosition()
    ghostStates = currentGameState.getGhostStates()
    scaredTimes = [ghostState.scaredTimer for ghostState in ghostStates]
    ghostLocations = currentGameState.getGhostPositions()
    food=currentGameState.getFood()#获取各项信息

    countWalls=0
    if walls[pacmanPosition[0]+1][pacmanPosition[1]]==True:
        countWalls=countWalls+1
    if walls[pacmanPosition[0]-1][pacmanPosition[1]]==True:
        countWalls=countWalls+1
    if walls[pacmanPosition[0]][pacmanPosition[1]+1]==True:
        countWalls=countWalls+1
    if walls[pacmanPosition[0]][pacmanPosition[1]-1]==True:
        countWalls=countWalls+1
    if countWalls==3:
        return -10000#如果是死胡同，返回一个很低的值
```

```
    dGhost=10000
    i=0
    for x in ghostLocations:
        if scaredTimes[i]<2:
            tem=abs(pacmanPosition[0]-x[0])+abs(pacmanPosition[1]-x[1])
            if tem<dGhost:
                dGhost=tem#计算最近的鬼的距离
        i=i+1
    min = 10000
    foodLocation = food.asList()
    min_x=0
    min_y=0
    for i in foodLocation:
        x = abs(pacmanPosition[0] - i[0])
        y = abs(pacmanPosition[1] - i[1])
        temp = x + y
        if min > temp:
            min = temp
            min_x=i[0]
            min_y=i[1]# 最近的豆子
    max=0
    for i in foodLocation:
        x = abs(min_x - i[0])
        y = abs(min_y - i[1])
        temp = x + y
        if max < temp:
            max = temp# 离最近豆子最远的豆子
    d=min+max#豆子距离的估计值
```

```
    d=min+max#豆子距离的估计值
    num = food.count()
    if dGhost <= 2:
        Evaluation = dGhost * 100
        if num == 0:
            Evaluation = Evaluation + 10
        else:
            Evaluation = Evaluation - num * 1 + 1.0 / d
        Evaluation = Evaluation - 10000
    else:
        if num == 0:
            Evaluation = 10000
        else:
            Evaluation = -num * 100 + 100.0 / d#同问题1的赋值

    return Evaluation+random.random()/10000+currentGameState.getScore()#防止吃掉某个豆子后不知道去那里
```

实现细节：①在判断胜利的返回值加上一个很小的随机数，使之在不影响其他的情况下每次都不相同，避免吃豆人在吃掉最后一个豆子后不知道去哪里从而停在最后一个豆子前

②判断吃豆人四个方向的墙的数量，如果是 3，则说明该处是一个胡同，为了使吃豆人不进入胡同，避免无意义的徘徊，我将吃豆人处于胡同处的状态赋值为一个较低的值

③由于可能有多个鬼，但是最近的一个鬼决定吃豆人的状态是否危险，所以取最近的鬼的距离来代表鬼的距离

④用：最近的豆子距离 min，距离该豆子最远的豆子两个豆子之间的距离 max，这两个值的和 min+max 来估计吃豆人吃完所有豆子的路径耗散

⑤评估值要加上当前的得分，防止吃豆人一直在某个地方徘徊

模块输入：GameState 的各种获取游戏状态信息的方法

数据结构定义：无

四． 实验结果

问题 1：

测试截图：

```
Pacman emerges victorious! Score: 1223
Pacman emerges victorious! Score: 1217
Pacman emerges victorious! Score: 1224
Pacman emerges victorious! Score: 1210
Pacman emerges victorious! Score: 1236
Pacman emerges victorious! Score: 1254
Pacman emerges victorious! Score: 1236
Pacman emerges victorious! Score: 1215
Pacman emerges victorious! Score: 1242
Pacman emerges victorious! Score: 1234
Average Score: 1229.1
Scores:        1223.0, 1217.0, 1224.0, 1210.0, 1236.0, 1254.0, 1236.0, 1215.0, 1242.0, 1234.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q1\grade-agent.test (4 of 4 points)
***     1229.1 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (0 of 0 points)
***         Grading scheme:
***          < 10:  fail
***         >= 10:  0 points
***     10 wins (2 of 2 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  0 points
***         >= 5:  1 points
***         >= 10:  2 points

### Question q1: 4/4 ###
```

描述说明：测试用例是将我所写的对行动的评估函数用于吃豆人的行为选择，共进行了 10 次测试，10 次测试的平均得分为 1229.1 分，胜利率为 10/10，并且十次游戏都没有出现超时的情况

问题 2：

测试截图：

```
*** 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
*** 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 ###
```

描述说明：测试了不同情况下我编写的 minimax 算法的表现，如子状态中包含成功状态和失败状态的情况；不同层数的情况；扩展的层数不同的情况；min 结点和 max 结点在不同层分布的情况等。其中包含有吃豆人失败的情况，但这是使用 minimax 算法的正常现象。我的最终得分为 84 分。

问题 3：

测试截图：

```
*** 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 ***
*** PASS: test_cases\q3\8-pacman-game.test

### Question q3: 5/5 ###
```

描述说明：在和 minimax 算法相同测试的基础上，本题的测试用例还检验了扩展的结点，用以判断是否进行了正确的剪枝，规避掉了不必要扩展的结点

问题 4：

测试截图：

```
*** PASS: test_cases\q4\0-eval-function-lose-states-1.test
*** PASS: test_cases\q4\0-eval-function-lose-states-2.test
*** PASS: test_cases\q4\0-eval-function-win-states-1.test
*** PASS: test_cases\q4\0-eval-function-win-states-2.test
*** PASS: test_cases\q4\0-expectimax1.test
*** PASS: test_cases\q4\1-expectimax2.test
*** PASS: test_cases\q4\2-one-ghost-3level.test
*** PASS: test_cases\q4\3-one-ghost-4level.test
*** PASS: test_cases\q4\4-two-ghosts-3level.test
*** PASS: test_cases\q4\5-two-ghosts-4level.test
*** PASS: test_cases\q4\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q4\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q4\6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent 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 ExpectimaxAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q4\7-pacman-game.test

### Question q4: 5/5 ###


Finished at 22:42:32

Provisional grades
==================
Question q4: 5/5
------------------
Total: 5/5
```

```
D:\人工智能作业\Project\Project_3>python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

D:\人工智能作业\Project\Project_3>python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Average Score: 15.0
Scores:        -502.0, 532.0, 532.0, 532.0, -502.0, -502.0, -502.0, -502.0, 532.0, 532.0
Win Rate:      5/10 (0.50)
Record:        Loss, Win, Win, Win, Loss, Loss, Loss, Loss, Win, Win
```

描述说明：本题将 ExpectimaxAgent 和 MinimaxAgent 相比较（第二张图），说明了在对面智能体并不总是选择最优行动的情况下这两种算法表现的差异，结果表明 ExpectimaxAgent 在这种情况下有时能够取得远远优于 MinimaxAgent 的结果

问题 5：

测试截图：

```
Pacman emerges victorious! Score: 893
Pacman emerges victorious! Score: 1158
Pacman emerges victorious! Score: 981
Pacman emerges victorious! Score: 804
Pacman emerges victorious! Score: 1171
Pacman emerges victorious! Score: 1174
Pacman emerges victorious! Score: 971
Pacman emerges victorious! Score: 1149
Pacman emerges victorious! Score: 963
Pacman emerges victorious! Score: 953
Average Score: 1021.7
Scores:        893.0, 1158.0, 981.0, 804.0, 1171.0, 1174.0, 971.0, 1149.0, 963.0, 953.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q5\grade-agent.test (6 of 6 points)
***     1021.7 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q5: 6/6 ###


Finished at 22:44:32

Provisional grades
==================
Question q5: 6/6
------------------
Total: 6/6
```

描述说明：测试用例是将我所写的对某个状态的评估函数用于吃豆人的行为选择，共进行了 10 次测试，10 次测试的平均得分为 1021.7 分，胜利率为 10/10，并且十次游戏都没有出现超时的情况

五． 总结与分析

1. 在 minimax 算法中，智能体将自己视为最大化自己的利益，而将对手视为最小化自己的利益。该算法在深度优先搜索的过程中逐步确定每个决策节点的值，从而确定最佳行动方案。
2. 然而，minimax 算法的计算复杂度很高，因此 alpha-beta 剪枝算法可以用来优化搜索过程。该算法通过评估子树中的最大和最小值来减少搜索时间。在实践中，alpha-beta 算法比 minimax 算法更有效，并且能够找到最佳决策方案。
3. 但是 minimax 算法和 alpha-beta 算法是基于对手也会做出最优选择的前提下的，当对手并不总是做出最优选择时，可能并不适用，因此可以根据对手行动的特点来具体的对状态进行评估，二不是总是评估为子节点的最小值

4. 关于评估函数的设计，既可以评估某个状态下智能体的行为，也可以直接评估某个状态，在本次实验中我总结了以下的编写评估函数的经验：
   ① 不同的因素在不同的状态下对状态的评估影响是不同的，比如当鬼离的较远时，鬼就几乎不影响游戏状态的评估，而当鬼离得较近时，则会成为评估的主要因素
   ② 可以适当的加入随机，以避免根据固定的评估值导致的停滞不前
   ③ 一些特殊的位置可以单独处理，比如本次实验中胡同的位置