گزارش پروژه دوم مبانی هوش مصنوعی

ایلیا نورانی ۹۹۳۱۰۹۲

۱)

قسمت اول :

از پارامتر newGhostStates برای چک کردن اینکه آیا عامل پکمن با روح در یک خانه قرار دارند یا خیر (اگر در یک خانه بودند و روح در حالت ترسیده نبود امتیاز -۹۰۰۰ بر می گرداند)استفاده می شود. به طور کلی می توان گفت برای محاسبه فاصله روح از عامل کاربرد دارد.

از پارامتر newfood و newPos برای محاسبه فاصله جانشین تا هر غذا و در نهایت محاسبه نزدیکترین غذا استفاده شده است. هر چقدر فاصله از غذا کمتر باشد بهتر است.

از پارامتر newScaredTimeزمانی استفاده می شود که بخواهیم فاصله عامل از روح را بدست بیاوریم . اگر روح در حالت ترسیده باشد ، هر چقدر این فاصله کمتر باشد بهتر است و اگر ترسیده نباشد ، بهتر است فاصله روح از عامل زیاد باشد.

تاثیر فاصله عامل از غذا نسبت به فاصله عامل از روح خیلی است. یعنی فاصله از روح می تواند باعث باخت عامل شود اما فاصله از غذا وقت تلف شده را افزایش می دهد که منجر کاهش امتیاز می شود.

قسمت دوم:

می توان از معکوس آن پارامتر استفاده کرد. به طور مثال فاصله از غذا هر چه کمتر باشد بهتر است.که یعنی نحوه تاپیر آن بر روی امتیاز به شکل زیر است:

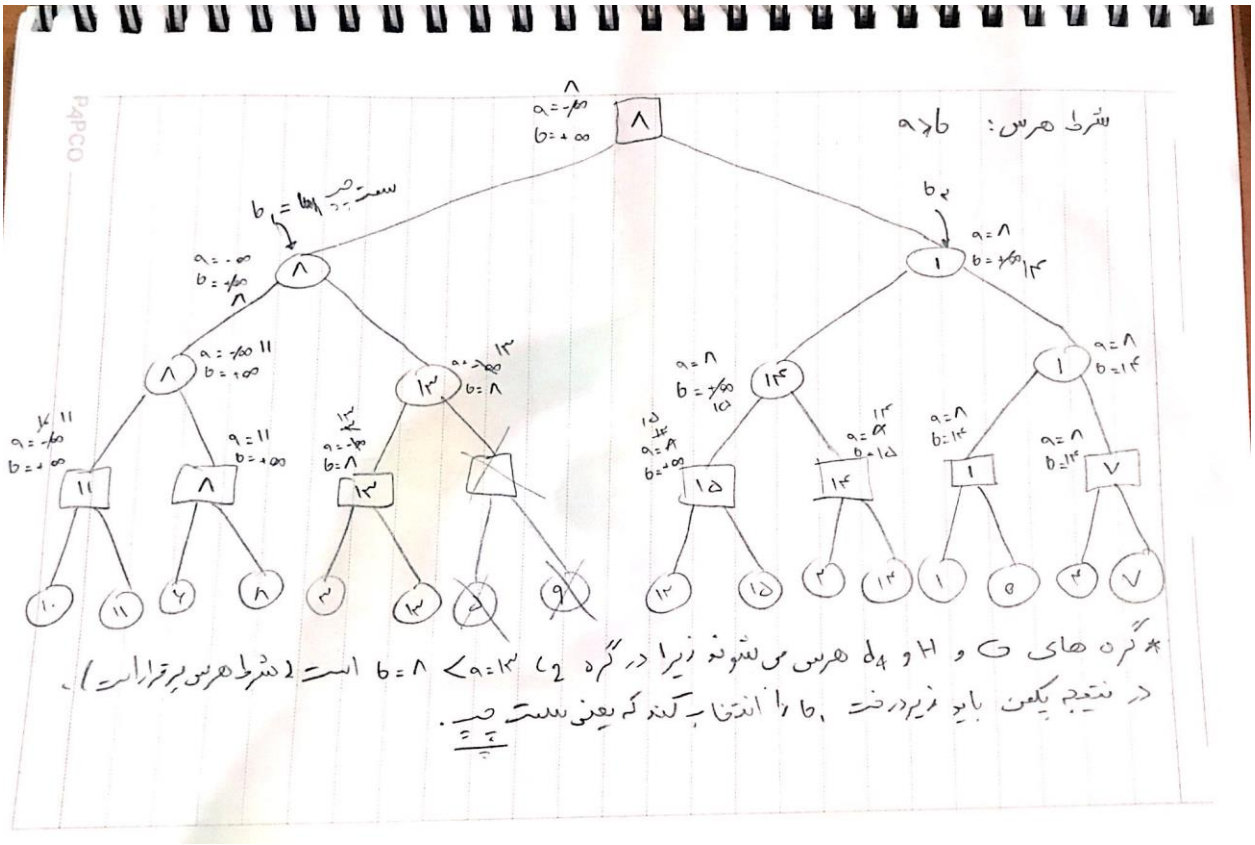
```
Score += 1 / foodDistance
```

و برای فاصله از روح که هرچقدر زیاد باشه ، می توانیم همان نسبت مستقیم را استفاده کنیم.

۲)

از آنجایی که پکمن می داند نمی تواند بازی را ببرد در نتیجه به این نتیجه می رسد که زنده ماندن آن فقط امتیاز او را کمتر می کند. در نتیجه در سریع ترین زمان ممکن خودش را به روح می رساند و می بازد تا کمترین مقدار امتیاز ممکن را از دست بدهد.

۳) بخش اول:



بخش دوم:

هرس آلفا بتا ممکن است در گره های میانی مقادیر متفاوتی با مقادیر بدست آمده بدون هرس تولید کند. دو پارامتر آلفا و بتا داریم که یکی برای حد بالا و دیگری برای حد پایین است . حال اگر حد پایین (آلفا) بزرگتر از حد بالا(بتا) شود دیگر لازم نیست جست و جو را ادامه دهیم و هرس می کنیم. حال از آنجایی که یک شاخه را هرس کردیم ، ممکن است مقدار خروجی زیر درختی که هرس شده است کمتر از مقداری باشد که عامل ما انتخاب کرده است(عامل مینیمم). اما از آنجایی

که می دانیم در سطح بالاتر مقدار این عامل انتخاب نمی شود ، در نتیجه برایمان مهم نیست که مقدار آن کمتر باشد. اما اگر از آلفا بتا استفاده نکنیم ، چون از پایین شروع به مقدار دهی می کنیم ، تمام مقادیر درست انتخاب می شوند که ممکن است با مقادیر الگوریتم آلفا بتا متفاوت باشند.اما این تفاوت فقط در گره های میانی است و امکان ندارد در ریشه رخ دهد. زیرا مقادیر آلفا بتا بر اساس عامل ریشه (مینیمم یا ماکسیمم) تغییر می کنند و در نهایت جواب یکسانی دارند. در واقع ریشه چون بالاتر از بقیه گره ها قرار دارد مقدار آن قطعا در هر دو الگوریتم یکسان می شود.

۴)

بخش اول:

در عملکرد مینیماکس عامل وقتی در دام قرار می گیرد خودش اقدام به باختن می کند زیرا می داند از آنجایی که روح ها باهوش عمل می کنند و بهترین حرکت ممکن را انجام می دهند(یعنی بدترین حالت برای پکمن) در نتیجه هیچ جوره نمی تواند برنده شود. اما در مینیماکس احتمالی عامل می داند ممکن است روح ها بهترین انتخاب را نکنند و مطابق میل پکمن حرکت کنند . در نتیجه اقدام به باختن نمی کند و احتمال برد دارد.

```
Record:       Loss

D:\AUT\AI\p2\AI-P2\AI-P2>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:\AUT\AI\p2\AI-P2\AI-P2>python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Average Score: -191.8
Scores:       532.0, -502.0, -502.0, -502.0, -502.0, -502.0, 532.0, -502.0, 532.0, -502.0
Win Rate:     3/10 (0.30)
Record:       Win, Loss, Loss, Loss, Loss, Loss, Win, Loss, Win, Loss

D:\AUT\AI\p2\AI-P2\AI-P2>
```

بخش دوم :

برای استفاده از رولت ویل در حالت excpectimax ، باید ارزیابی حالتی که بعد از انجام هر اکشن به جود می آید را انجام دهیم (در تابع expValue) و با توجه به مقدار آن یک احتمال از پیش تعیین شده برای آن در نظر می گیریم و در نهایت به صورت رندوم یکی را انتخاب می کنیم.

حال اگر نیاز بود تا بیشتر از یک حالت انتخاب شود ، به ازای هر حرکت مجاز یکبار تابع expValue صدا زده می شود و در نهایت بین مقادیر آن ها ماکسیمم گرفته می شود.

۵)

در این تابع ارزیابی علاوه بر در نظر گرفتن فاصله از غذا و فاصله از روح ها ، فاصله از کپسول ها (غذا هایی که با خوردن آن پکمن می تواند روح ها را بخورد) و همچنین فاصله از روح در حالتی که کپسول را خورده است مورد بررسی قرار می گیرد. از آنجایی که ویژگی های بیشتری در این تابع ارزیابی بررسی می شود ، این تابع ارزیابی دقیق تری نسبت به تابع بخش اول دارد و در نتیجه عملکرد بهتری خواهد داشت.

توضیح کد:

تابع evaluationFunction :

چهار متغیر successorGameState,newFoods,newGhostStates,newScaredTimes به علاوه موقعیت مکانی پکمن را ذخیره می کنیم. در خط های ۷۹ تا ۸۵ چک می کنیم پکمن برنده شده است یا خیر و اگر پکمن موقیعتیش با روح یکسان بود در حالی که روح در حالت عادی بود ، یعنی پکمن بازنده شده است و امتیاز مربوطه را بر می گرداند. در خطوط ۸۶ تا ۹۷ ، فاصله پکمن تا روح ها را محاسبه می کنیم و نزدیک ترین غذا به آن را مشخص می کنیم(نحوه تغییر امتیاز در بخش سوال مربوط به قسمت توضیح داده شده است). در بخش آخر این تابع ، فاصله از روح را در دو حالت فعلی و حالت بعدی محاسبه می کنیم . سپس بررسی می کنیم اگر روح در حالت عادی بود این فاصله باید زیاد باشد و در غیر اینصورت بهتر است فاصله کم باشد تا پکمن بتواند روح را شکار کند.با توجه به شرایط امتیاز مربوطه داده می شود. در نهایت مقدار امتیاز برآورد شده بازگردانده می شود. کد مربوط به این بخش در ادامه آمده است:

```
51
52          return legalMoves[chosenIndex]
53
54      def evaluationFunction(self, currentGameState, action):
55          """
56          Design a better evaluation function here.
57
58          The evaluation function takes in the current and proposed successor
59          GameStates (pacman.py) and returns a number, where higher numbers are better.
60
61          The code below extracts some useful information from the state, like the
62          remaining food (newFood) and Pacman position after moving (newPos).
63          newScaredTimes holds the number of moves that each ghost will remain
64          scared because of Pacman having eaten a power pellet.
65
66          Print out these variables to see what you're getting, then combine them
67          to create a masterful evaluation function.
68          """
69          # Useful information you can extract from a GameState (pacman.py)
70          successorGameState = currentGameState.generatePacmanSuccessor(action)
71          newPos = successorGameState.getPacmanPosition()
72          newFood = successorGameState.getFood().asList()
73          newGhostStates = successorGameState.getGhostStates()
74          newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
75
76          "*** YOUR CODE HERE ***"
77
78          pacmanPositionition = currentGameState.getPacmanPosition()
79
80          if successorGameState.isWin():
81              return 90000
82
83          for ghost_state in newGhostStates:
84              if ghost_state.getPosition() == pacmanPositionition and ghost_state.scaredTimer == 0:
85                  return -90000
86
87          score = 0
88
89          if action == 'Stop':
90              score -= 100
91
92          foodDist = [util.manhattanDistance(newPos, food) \
93          for food in newFood]
94          nearestFood = min(foodDist)
95          score += float(1/nearestFood)
96          score -= len(newFood)
97
98          currentGhostDist = [util.manhattanDistance(newPos, ghost.getPosition()) \
99          for ghost in currentGameState.getGhostStates()]
```

```
        newGhostStates = successorGameState.getGhostStates()
        newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

        "*** YOUR CODE HERE ***"

        pacmanPositionition = currentGameState.getPacmanPosition()

        if successorGameState.isWin():
            return 90000

        for ghost_state in newGhostStates:
            if ghost_state.getPosition() == pacmanPositionition and ghost_state.scaredTimer == 0:
                return -90000

        score = 0

        if action == 'Stop':
            score -= 100

        foodDist = [util.manhattanDistance(newPos, food) \
        for food in newFood]
        nearestFood = min(foodDist)
        score += float(1/nearestFood)
        score -= len(newFood)

        currentGhostDist = [util.manhattanDistance(newPos, ghost.getPosition()) \
        for ghost in currentGameState.getGhostStates()]
        nearestCurrentGhost = min(currentGhostDist)

        newGhostDist = [util.manhattanDistance(newPos, ghost.getPosition()) \
        for ghost in newGhostStates]
        nearestNewGhost = min(newGhostDist)

        sumScaredTimes = sum(newScaredTimes)
        if sumScaredTimes > 0 :
            if nearestNewGhost < nearestCurrentGhost:
                score += 200
            else:
                score -= 100
        else:
            if nearestNewGhost < nearestCurrentGhost:
                score -= 100
            else:
                score += 200

        return successorGameState.getScore() + score

def scoreEvaluationFunction(currentGameState):
    """
```

کلاس MinimaxAgent :

این کلاس برای پیاده سازی درخت مینیماکس استفاده می شود. تابع minvalue برای عامل مینیمم استفاده می شود. به این صورت که ابتدا بررسی می کند اگر عامل حرکت مجاز نداشت ، تابع ارزیابی آ را صدا میزند . در غیر این صورت طبق فرمول مقدار مینیمم در درخت مینیماکس برای هر عامل این مقدار را بدست می آورد(به ازای تمام حرکت ها). تابع

maxValue که فقط برای عامل پکمن استفاده می شود ، مانند تابع minValue است با این تفاوت که مقدار ماکسیمم را محاسبه می کند. در نهایت از تمام حالت های مینیمم که انجام شده است ، ماکسیمم می گیرد و مقدار آن را باز می گرداند. تمام این کد های این بخش در تابع getAction پیاده سازی شده است . کد این بخش در ادامه آمده است.

```
class MinimaxAgent(MultiAgentSearchAgent):
    """
    Your minimax agent (question 2)
    """

    def getAction(self, 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 ***"

        def minValue(state, agentIndex, depth):

            agentCount = gameState.getNumAgents()
            legalActions = state.getLegalActions(agentIndex)

            if not legalActions:
                return self.evaluationFunction(state)

            if agentIndex == agentCount - 1:
                minimumValue =  min(maxValue(state.generateSuccessor(agentIndex, action), \
                agentIndex,  depth) for action in legalActions)
            else:
                minimumValue = min(minValue(state.generateSuccessor(agentIndex, action), \
                agentIndex + 1, depth) for action in legalActions)

            return minimumValue

        def maxValue(state, agentIndex, depth):
```

```
177         """
178         "*** YOUR CODE HERE ***"
179
180         def minValue(state, agentIndex, depth):
181
182             agentCount = gameState.getNumAgents()
183             legalActions = state.getLegalActions(agentIndex)
184
185             if not legalActions:
186                 return self.evaluationFunction(state)
187
188             if agentIndex == agentCount - 1:
189                 minimumValue =  min(maxValue(state.generateSuccessor(agentIndex, action), \
190                 agentIndex,  depth) for action in legalActions)
191             else:
192                 minimumValue = min(minValue(state.generateSuccessor(agentIndex, action), \
193                 agentIndex + 1, depth) for action in legalActions)
194
195             return minimumValue
196
197         def maxValue(state, agentIndex, depth):
198
199             agentIndex = 0
200             legalActions = state.getLegalActions(agentIndex)
201
202             if not legalActions  or depth == self.depth:
203                 return self.evaluationFunction(state)
204
205             maximumValue =  max(minValue(state.generateSuccessor(agentIndex, action), \
206             agentIndex + 1, depth + 1) for action in legalActions)
207
208             return maximumValue
209
210         actions = gameState.getLegalActions(0)
211
212         allActions = {}
213         for action in actions:
214             allActions[action] = minValue(gameState.generateSuccessor(0, action), 1, 1)
215
216         return max(allActions, key=allActions.get)
217
218         util.raiseNotDefined()
```

کلاس AlphaBetaAgent :

یک تابع minValue دارد که مانند قسمت قبل است با این تفاوت که در ورودی یک آلفا و بتا هم می گیرد و الگوریتم آلفا بتا را پیاده سازی می کند.در خط ۲۴۱ تا ۲۵۷ چک کردن آلفا قابل مشاهده است(زیرا minvalue است). تابع maxValue نیز مانند بخش های قبل است با این تفاوت که آلفا بتا دارد و مقدار بتا با مقدار ماکسیمم مقایسه می شود(خطوط ۲۷۱ تا ۲۷۹).در نهایت هم از این دو تابع استفاده می شود و الگوریتم آلفا بتا کامل می شود(کد در ادامه...).

```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning (question 3)
    """
    def getAction(self, gameState):
        """
        Returns the minimax action using self.depth and self.evaluationFunction
        """
        "*** YOUR CODE HERE ***"

        def minValue(state, agentIndex, depth, alpha, beta):

            agentCount = gameState.getNumAgents()
            legalActions = state.getLegalActions(agentIndex)

            if not legalActions:
                return self.evaluationFunction(state)

            minimumValue = 99999
            currentBeta = beta

            if agentIndex == agentCount - 1:
                for action in legalActions:
                    minimumValue =  min(minimumValue, maxValue(state.generateSuccessor(agentIndex, action), \
                    agentIndex,  depth, alpha, currentBeta))
                    if minimumValue < alpha:
                        return minimumValue
                    currentBeta = min(currentBeta, minimumValue)

            else:
                for action in legalActions:
                    minimumValue =  min(minimumValue,minValue(state.generateSuccessor(agentIndex, action), \
                    agentIndex + 1, depth, alpha, currentBeta))
                    if minimumValue < alpha:
                        return minimumValue
                    currentBeta = min(currentBeta, minimumValue)

            return minimumValue

        def maxValue(state, agentIndex, depth, alpha, beta):

            agentIndex = 0
            legalActions = state.getLegalActions(agentIndex)

            if not legalActions  or depth == self.depth:
                return self.evaluationFunction(state)

            maximumValue = -99999
```

```
                    if minimumValue < alpha:
                        return minimumValue
                    currentBeta = min(currentBeta, minimumValue)

            else:
                for action in legalActions:
                    minimumValue =  min(minimumValue,minValue(state.generateSuccessor(agentIndex, action), \
                    agentIndex + 1, depth, alpha, currentBeta))
                    if minimumValue < alpha:
                        return minimumValue
                    currentBeta = min(currentBeta, minimumValue)

            return minimumValue

        def maxValue(state, agentIndex, depth, alpha, beta):

            agentIndex = 0
            legalActions = state.getLegalActions(agentIndex)

            if not legalActions  or depth == self.depth:
                return self.evaluationFunction(state)

            maximumValue = -99999
            currentAlpha = alpha

            for action in legalActions:
                maximumValue = max(maximumValue, minValue(state.generateSuccessor(agentIndex, action), \
                agentIndex + 1, depth + 1, currentAlpha, beta) )
                if maximumValue > beta:
                    return maximumValue
                currentAlpha = max(currentAlpha, maximumValue)
            return maximumValue

        actions = gameState.getLegalActions(0)
        alpha = -99999
        beta = 99999

        allActions = {}
        for action in actions:
            value = minValue(gameState.generateSuccessor(0, action), 1, 1, alpha, beta)
            allActions[action] = value

            if value > beta:
                return action
            alpha = max(value, alpha)

        return max(allActions, key=allActions.get)
        util.raiseNotDefined()
```

کلاس ExpectimaxAgent :

مانند کلاس MinimaxAgent است فقط به جای تابع minValue تابع expValue دارد. این تابع یک مقدار احتمال دارد(probability) که در هر مقدار مورد انتظاری که توسط این تابع محاسبه می شود ، ضرب می شود. بقیه قسمت های این تابع مشابه MinimaxAgent است تنها تابع expValue جایگزین تابع minValue شده است.

کد این بخش نیز در ادامه آمده است.

```python
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (question 4)
    """
    def getAction(self, 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 ***"
        def expValue(state, agentIndex, depth):
            agentCount = gameState.getNumAgents()
            legalActions = state.getLegalActions(agentIndex)
            if not legalActions:
                return self.evaluationFunction(state)

            expectedValue = 0
            probabilty = 1.0 / len(legalActions)
            for action in legalActions:
                if agentIndex == agentCount - 1:
                    currentExpValue =  maxValue(state.generateSuccessor(agentIndex, action), \
                    agentIndex,  depth)
                else:
                    currentExpValue = expValue(state.generateSuccessor(agentIndex, action), \
                    agentIndex + 1, depth)
                expectedValue += currentExpValue * probabilty

            return expectedValue


        def maxValue(state, agentIndex, depth):

            agentIndex = 0
            legalActions = state.getLegalActions(agentIndex)

            if not legalActions  or depth == self.depth:
                return self.evaluationFunction(state)

            maximumValue =  max(expValue(state.generateSuccessor(agentIndex, action), \
            agentIndex + 1, depth + 1) for action in legalActions)

            return maximumValue
        actions = gameState.getLegalActions(0)
        allActions = {}
        for action in actions:
            allActions[action] = expValue(gameState.generateSuccessor(0, action), 1, 1)
        return max(allActions, key=allActions.get)
        util.raiseNotDefined()
```

تابع betterEvaluationFunction :

مانند تابع evaluationFunction است بای این تفاوت که دو ویژگی فاصله از کپسول و فاصله از روح در حالت شکار را نیز بررسی می کند . در خطوط ۳۹۵ تا ۴۰۱ فاصله از کپسول و از خط ۴۰۱ تا ۴۱۱ فاصله از روح در حالت شکار مورد بررسی قرار می گیرد. به طور کلی هر جا که مقدار بیشتر برایمان مفید باشد ، از نسبت مستقیم و در غیر این صورت از نسب معکوس استفاده می کنیم . کد این بخش در ادامه آمده است:

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

    Don't forget to use pacmanPosition, foods, scaredTimers, ghostPositions!
    DESCRIPTION: <write something here so we know what you did>
    """
    "*** YOUR CODE HERE ***"
    pacmanPosition = currentGameState.getPacmanPosition()
    foods = currentGameState.getFood().asList()
    ghostStates = currentGameState.getGhostStates()
    scaredTimers = [ghostState.scaredTimer for ghostState in ghostStates]
    ghostPositions = currentGameState.getGhostPositions()
    #ghostPosition = ghost.getPosition
    currentCapsule = currentGameState.getCapsules()
    if currentGameState.isWin():
        return 90000

    for state in ghostStates:
        if state.getPosition() == pacmanPosition and state.scaredTimer == 1:
            return -90000

    score = 0

    foodDistance = [util.manhattanDistance(pacmanPosition, food) \
    for food in foods]
    nearestFood = min(foodDistance)
    score += float(1/nearestFood)
    score -= len(foods)

    if currentCapsule:
        capsuleDistance = [util.manhattanDistance(pacmanPosition, capsule) \
        for capsule in currentCapsule]
        nearestCapsule = min(capsuleDistance)
        score += float(1/nearestCapsule)

    currentGhostDistances = [util.manhattanDistance(pacmanPosition, ghost.getPosition()) \
    for ghost in currentGameState.getGhostStates()]
    nearestCurrentGhost = min(currentGhostDistances)
    scaredTime = sum(scaredTimers)
    if nearestCurrentGhost >= 1:
        if scaredTime < 0:
            score -= 1/nearestCurrentGhost
        else:
            score += 1/nearestCurrentGhost

    return currentGameState.getScore() + score

```