به نام خدا

درس : مبانی و کاربردهای هوش مصنوعی

استاد: جوانمردی

دانشجو: مهدی منصوری خواه

شماره دانشجویی: 9931056

گزارش پروژه – دوم(پکمن)

عامل عکس العمل

در این بخش باید تابع evaluationFunction کامل شود .

def evaluationFunction(self, currentGameState, action)

```
successorGameState = currentGameState.generatePacmanSuccessor(action)
newPos = successorGameState.getPacmanPosition()
newFood = successorGameState.getFood()
newGhostStates = successorGameState.getGhostStates()
newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

"*** YOUR CODE HERE ***"
newCapsule = successorGameState.getCapsules()
foodScore, minDistance = foodScoreEstimate(newFood, newPos)
capsuleScore = capsuleScoreEstimate(newCapsule, minDistance, newPos)
score = successorGameState.getScore()
postion = positionEstimate(currentGameState.getPacmanPosition(), newPos=newPos)
ghostScore = ghostStateEstimate(newGhostStates, newPos, minDistance)
scaredScore = scaredScoreEstimate(newGhostStates, minDistance, newPos)
scaredTimer = sum(newScaredTimes)
return postion + (ghostScore * 1.42) + (scaredTimer * 1.32) + (score * 1.10) + (foodScore * 1.45 ) + (scaredScore * 1.85) + (capsuleScore * 0.4)
```

تا قبل از اینکه عامل از function evaluation جدید استفاده کند، صرفا بر اساس حالت خروجی ارزیابی خود را انجام میداد .اما با استفاده از تابع ارزیابی جدیدی که تعریف شد، هم بر اساس action و همچنینstate ثانویه ارزیابی انجام میشود .این تابع ارزیابی، با استفاده از ترکیب خطی وزن دار چندین علت مختلف ، میتواند بهتر عمل کند .برای مثال به ازای هر غذایی که در نزدیکی پکمن قرار دارد، میتوان مقدار ارزیابی را افزایش داد چون پکمن امیدوار تر است تا بازی را ببرد و امتیازش بیشتر شود .عنصر مهم دیگر برا ارزیابی دقیق، این است که روح ها در چه فاصله ای نسبت به پکمن قرار دارند .اگر فاصله روح و پک من صفر باشد(دریک خانه باشند) باید وزن منفی نسبتا زیادی داد تا پکمن بفهمد که در حال باختن است. همچنین هرچه روح ها به پکمن نزدیک تر میشوند، پکمن باید احساس خطر کند .نهایتا اینکه هرچه روح ها زمان بیشتری در حالت ترس باشند، پکمن با خیال راحتتری حرکت میکند پس مقدار ارزیابی باید افزایش یابد .ترکیب وزن دار این علت ها و نیز امتیاز state نهایی، به عنوان خروجی تابع ارزیابی در نظر گرفته می شود.(این اعداد به صورت تجربی بدست آمده اند تا پکمن بتواند بهترین حرکت خود را انجام بدهد و همین علت این اعداد خیلی دقیق نیستند و می توان آن را بهینه تر کرد)

توابع به صورت زیر پیاده سازی شده اند همانطور که گفته شد امتیازها و ضرایب دقیق نیستند و میتوان امتیاز و ضرایب بهتری به آن ها اختصاص داد در اینجا این امتیاز ها و ضرایب مناسب هستند و هم در نقشه testclassic و هم در نقشه mediumClassic با یک روح به خوبی کار میکنند

Position : امتیاز منفی برای عدم حرکت عامل

time Scared: امتیاز مثبت بابت زمان باقی مانده ترسیدن روح ها

Score : امتیاز کسب شده در حرکت بعدی

score Scared: امتیاز مثبت برخورد به روح های ترسیده

score Food : امتیاز فاصله نزدیک ترین نقطه

score Ghost : امتیاز منفی نزدیک شدن روح

score Capsules: امتیاز مثبت برای خوردن کپسول

```python
def foodScoreEstimate(newFood, NewPositions):
    foodDistance = []
    for i in newFood.asList():
      if newFood[i[0]][i[1]]:
        foodDistance.append(manhattanDistance(NewPositions, i))
    foodScore = 0
    minDistance = 0
    if foodDistance:
        minDistance = min(foodDistance)
        if minDistance == 0:
            foodScore = 100
        else:
            foodScore = 10/minDistance
    return foodScore, minDistance
```

```python
def ghostStateEstimate(newGhostStates, newPos, minDistance):
    activeGhostPositions = [i.getPosition() for i in newGhostStates if i.scaredTimer == 0]
    ghostScore = 100
    maxGhostDistance = -1
    minGhostDistance = -1
    if newPos in activeGhostPositions:
        ghostScore = -1000
    activeGhostDistance = [manhattanDistance(newPos, i) for i in activeGhostPositions]

    if activeGhostDistance:
        maxGhostDistance = max(activeGhostDistance)
        minGhostDistance = min(activeGhostDistance)
        if minGhostDistance <= 1:
            ghostScore = -1000
            minGhostDistance = -1
        else:
            if minGhostDistance < minDistance:
                ghostScore = -100
            ghostScore = maxGhostDistance
    return ghostScore
```

```python
def capsuleScoreEstimate(newCapsule, minDistance, newPos):
    capsuleScore = 0
    if len(newCapsule) >= 1:
        capsuleDistance = min([manhattanDistance(newPos, i) for i in newCapsule])
    if newPos in newCapsule and capsuleDistance < minDistance:
        capsuleScore = 2000
    return capsuleScore

def scaredScoreEstimate(newGhostStates, minDistance, newPos):
    scaredGhostPositions = [i.getPosition() for i in newGhostStates if i.scaredTimer != 0]
    scaredScore = 0
    if len(scaredGhostPositions) > 0:
        if newPos in scaredGhostPositions:
            scaredScore = 20000
        else:
            scaredGhostDistance = min([manhattanDistance(newPos, i) for i in scaredGhostPositions])
            if scaredGhostDistance < minDistance:
                scaredScore = minDistance - scaredGhostDistance
    return scaredScore

def positionEstimate(currentPos, newPos):
    if currentPos == newPos:
        return -100
    else:
        return 0
```

سوال: توضیح دهید که از هرکدام از پارامترهای دخیل در تابع ارزیابی چگونه استفاده کرده اید و هر کدام چگونه بر روی خروجی تاثیر میگذارند؟

توضیح قسمت اول دربالا آمده است و برای تکمیل آن میتوان گفت مثلا برای پیدا کردن نزدیک ترین نقطه از فاصله منهتن استفاده شده است و برای فاصله عامل ما تا روح ها یک مقدار منفی داده شده است که اگر در ادامه ،عامل ما با روح برخورد داشته باشد امتیاز

1000- میگیرد. همانطور که در بالاتر مشاهده کردید امتیاز برخورد روح ترسیده و فاصله از روح ها بیشترین تأثیر را دارد. علت اصلی آن هم دوری کردن عامل از شکست خوردن است؛ عامل ترجیح میدهد زمان بیشتری را بگذراند و دیرتر به هدف برسد تا اینکه زودتر شکست بخورد و متوقف شود.

سوال: چگونه میتوان پارامترهایی که در یک راستا نمی باشند را با یکدیگربرای تابع ارزیابی ترکیب کرد؟

برای محاسبه تابع ارزیابی، پارامترهای مثبت و منفی مختلفی وجود دارد که هر کدام به نوعی به روی یکدیگر تأثیر می گذارند که در قسمت بالا هم به آن اشاره شد مثلا برای برخورد عامل به روح، امتیاز منفی -1000 در نظر گرفته میشود از طرفی دیگر خوردن کپسول امتیاز مثبت دارد.

در نتیجه با دادن امتیاز مثبت و منفی به پارامترها می توان آن ها تحلیل کرد.

مینیماکس

```
"*** YOUR CODE HERE ***"
import sys
num_agents = gameState.getNumAgents()
def max_value(index, state, depth):
    if state.isWin() or state.isLose() or depth + 1 == self.depth:
        return self.evaluationFunction(state)
    v = -sys.maxsize
    legalActions = state.getLegalActions(index)
    ai = (index + 1) % num_agents
    for action in legalActions:
        successor = state.generateSuccessor(index, action)
        v = max(v, min_value(ai, successor, depth + 1))
    return v
```

```
def min_value(index, state, depth):
    if state.isWin() or state.isLose():
        return self.evaluationFunction(state)
    v = sys.maxsize
    legalActions = state.getLegalActions(index)
    ai = (index + 1) % num_agents
    if ai == 0:
        for action in legalActions:
            successor = state.generateSuccessor(index, action)
            v = min(v, max_value(ai, successor, depth))
    else:
        for action in legalActions:
            successor = state.generateSuccessor(index, action)
            v = min(v, min_value(ai, successor, depth))
    return v

legalActions = gameState.getLegalActions(0)
act = max(legalActions, key=lambda action: min_value(1, gameState.generateSuccessor(0, action), 0))
return act
# util.raiseNotDefined()
```

def getAction(self, gameState)

در این تابع برای لایه های مربوط به min value یک تابع داریم و برای max value هم یک تابع داریم که باگرفتن state فعلی agent و شماره agent وگرفتن عمقی که در آن قرار است عملیات پیدا کردن min value یا max value صورت بگیرد مقدار value را مشخص می کند.

تابع max_value :

در پیاده سازی این تابع از شبه کد موجود در اسالاید های درس استفاده شده است به این صورت که ابتدا مقدار Value که با V نمایش داده شده است برابر منفی بینهایت قرار می دهیم، سپس حرکت های مجازی که میتوان انجام داد توسط عامل را بدست می آوریم و ماکزیمم بین value فعلی و value که پس از پیدا کردن min value از عمق بعدی بدست می آید به عنوان value در نظر گرفته می شود.

تابع min_value هم شبیه به تابع بالا پیاده سازی می شود.

```
legalActions = gameState.getLegalActions(0)

act = max(legalActions, key=lambda action: min_value(1, gameState.generateSuccessor(0, action), 0))

return act
```

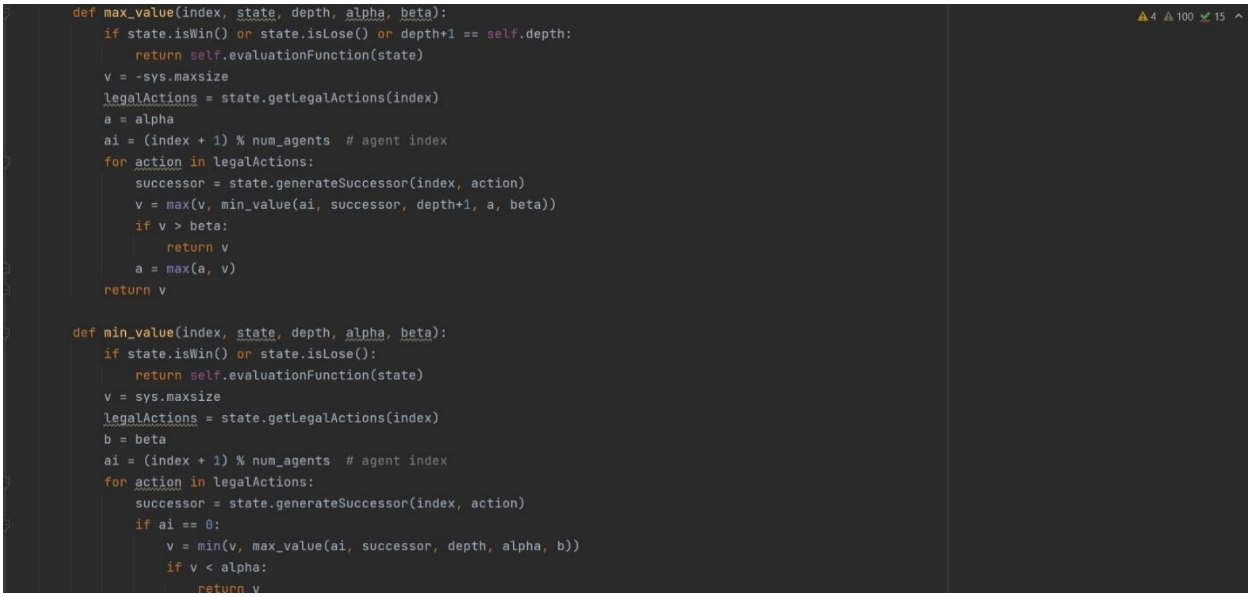
در این قسمت ،ازاکشن های موجود بهترین اکشن انتخاب می شود.

سوال: بررسی کنید چرا پکمن در این حالت به دنبال باخت سریع تر است؟

چون پکمن میداند که روح ها بهترین بازی خود را انجام میدهند و اگر این اتفاق بیفتد با توجه به نقشه و جایگاه عامل ها، حتما پکمن بدون خوردن غذا می بازد برای همین ترجیح می دهد که با بالاترین امتیاز ممکن بازی را خاتمه دهد پس سریعتر خود را به کشتن میدهد تا امتیاز منفی نگیرد.

هرس آلفا-بتا

پیاده سازی این بخش شبیه بخش قبل می باشد با این تفاوت که مقدار beta , alpha داریم که set می شوند.



```
def max_value(index, state, depth, alpha, beta):
    if state.isWin() or state.isLose() or depth+1 == self.depth:
        return self.evaluationFunction(state)
    v = -sys.maxsize
    legalActions = state.getLegalActions(index)
    a = alpha
    ai = (index + 1) % num_agents  # agent index
    for action in legalActions:
        successor = state.generateSuccessor(index, action)
        v = max(v, min_value(ai, successor, depth+1, a, beta))
        if v > beta:
            return v
        a = max(a, v)
    return v

def min_value(index, state, depth, alpha, beta):
    if state.isWin() or state.isLose():
        return self.evaluationFunction(state)
    v = sys.maxsize
    legalActions = state.getLegalActions(index)
    b = beta
    ai = (index + 1) % num_agents  # agent index
    for action in legalActions:
        successor = state.generateSuccessor(index, action)
        if ai == 0:
            v = min(v, max_value(ai, successor, depth, alpha, b))
            if v < alpha:
                return v
            b = min(b, v)
        else:
            v = min(v, min_value(ai, successor, depth, alpha, b))
            if v < alpha:
                return v
            b = min(b, v)
    return v

alpha = -sys.maxsize
beta = sys.maxsize
score = -sys.maxsize
act = None
legalActions = gameState.getLegalActions(0)
for action in legalActions:
    state = gameState.generateSuccessor(0, action)
    if min_value(1, state, 0, alpha, beta) > score:
        act = action
        score = min_value(1, state, 0, alpha, beta)
    if score > beta:
        return act
    alpha = max(alpha, score)
return act
# util.raiseNotDefined()
```

تابع max_value:

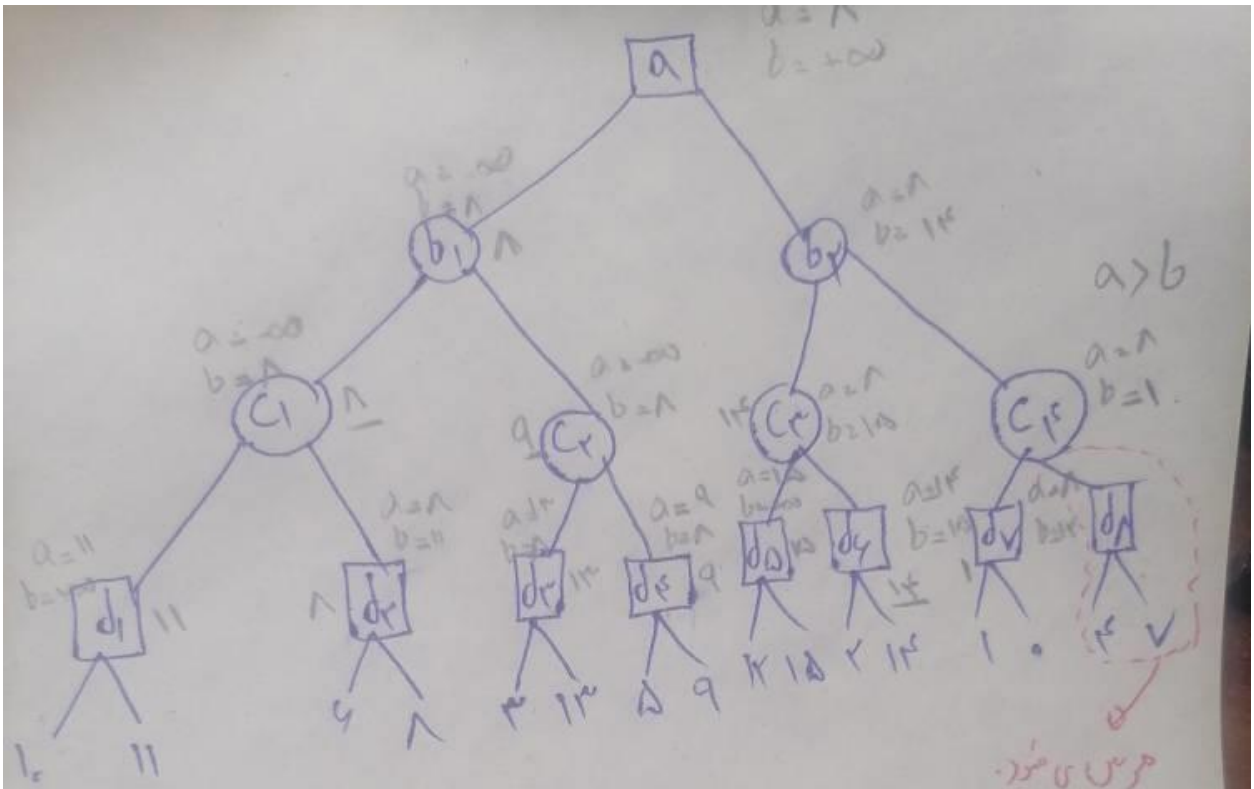
تنها تغییری که اعمال شده است این است که هر بار که اکشن انجام می دهیم باید چک کنیم که آیا مقدار value از بتا بالاتر رفته است که هرس انجام بدیم یا خیر و آلفا هم هر دفعه با value مقایسه می شود و مینیمم گرفته میشود.

تابع min_value :

این تابع شبیه تابع بالایی کار میکند که باید بررسی کنیم که حالت بعدی min هست یا max و هرگاه مقدار value از آلفا کمتر شد هرس را انجام دهیم و بتا هم هر دفعه با مقدار value باید مینیمم گرفته شود.

و در آخر هم بهترین اکشن را انتخاب می کنیم. دقیقا مثل روشی است که در کلاس انجام میدادیم مقدار آلفا را برابر منفی بی نهایت و مقدار بتا را برابر مثبت بی نهایت قرار میدهیم و هر دفعه مقادیر آلفا و بتا را آپدیت می کنیم و شرط هرس(الفا>=بتا) را چک میکنیم و با for زدن روی اکشن های مجاز بهترین اکشن رو انتخاب می کنیم.

سوال: فرض کنید درخت زیر بیک از تستهای داده شده به الگوریتم آلفا-بتا شما است. گرههای مربوط به پکمن با مرب ع و گرههای هر روح با دایره نمایش داده شده است. در وضعیت فعیل پکمن دو حرکت مجاز دارد، یا یمتواند به سمت راست حرکت کرده و وارد زیر درخت b 2شود و یا به سمت چپ حرکت کرده و وارد زیر درخت b 1شود. الگوریتم آلفا-بتا را تا عمق ۴ روی درخت زیر شوند. همچن یی اجرا کرده و مشخص کنید کدام گرهها و به چه دلیل هرس یم مشخص کنید در وضعیت فعیل، حرکت بعدی پکمن باید به سمت راست باشد یا چپ؟


a
b1
b2
C1
C2
C14
d1
d2
d3
d4
d5
d6
d7
d8
a>b
هرس می شود.

در شاخه سمت راست به دلیل اینکه شرط هرس برقرار است(الفا=>بتا) قسمت مشخص شده هرس می شود و نیاز به پیمایش کردن نمی باشد به همین علت باید حرکت بعدی پکمن به سمت راست باشد که تعداد گره های کمتری را بررسی کند.

سوال: آیا در حالتکیل هرس آلفا-بتا قادر است که مقداری متفاوت با مقدار به دست آمده بدون هرس را در ریشه درخت تولید کند؟ در گره های میانی چطور؟ به طور خالصه دلیل خودتان را توضیح دهید.

هرس کردن برای کمتر پیمایش کردن درخت می باشد که باعث می شود کل شاخه ها را بررسی نکنیم در این حین ممکن است در گره های میانی مقداری نادرست و متفاوت از آن چیزی که انتظارش را داریم تولید کند که اهمیت چندانی ندارد چون در نهایت جواب در ریشه به درستی نشان داده می شود. این تفاوت در شرط هرس کردن(الفا>=بتا) به وجود می آید چون قطعا گره در ادامه میتواند مقدار کوچکتر یا بزرگتر از آن را جایگزینش کند و بررسی آن سودی ندارد مثل سوال قبل که قسمتی از درخت هرس شده است.

مینیماکس احتمالی

```
    agentsNum = gameState.getNumAgents()

    def isTerminalState(gameState, currentDepth, agentIndex):
        return currentDepth == self.depth or \
               not gameState.getLegalActions(agentIndex)

    def maxValue(gameState, currentDepth, agentIndex):
        bestResult = [-999999]

        if isTerminalState(gameState, currentDepth, agentIndex):
            bestResult[0] = self.evaluationFunction(gameState)
            return bestResult

        for a in gameState.getLegalActions(agentIndex=agentIndex):
            v = bestResult[0]
            nextResult = expValue(gameState.generateSuccessor(agentIndex=agentIndex, action=a), currentDepth,
                                  agentIndex=1)
            next_v = nextResult[0]
            if v < next_v:
                bestResult[0] = next_v
                try:
                    bestResult[1] = a
                except IndexError:  # for the first iteration, we haven't pushed the action before
                    bestResult.append(a)

        return bestResult
```

```python
def expValue(gameState, currentDepth, agentIndex):
    bestResult = [0]

    if isTerminalState(gameState, currentDepth, agentIndex):
        bestResult[0] = self.evaluationFunction(gameState)
        return bestResult

    if agentIndex < agentsNum - 1:  # then we should determine other ghosts' decision
        for a in gameState.getLegalActions(agentIndex=agentIndex):
            p = 1. / len(gameState.getLegalActions(agentIndex=agentIndex))
            v = expValue(gameState.generateSuccessor(agentIndex=agentIndex, action=a), currentDepth,
                         agentIndex=agentIndex + 1)
            bestResult[0] += p * v[0]

    else:  # the next agent is pacman
        for a in gameState.getLegalActions(agentIndex=agentIndex):
            p = 1. / len(gameState.getLegalActions(agentIndex=agentIndex))
            v = maxValue(gameState.generateSuccessor(agentIndex=agentIndex, action=a), currentDepth + 1,
                         agentIndex=0)
            bestResult[0] += p * v[0]

    return bestResult

return maxValue(gameState, currentDepth=0, agentIndex=0)[1]  # second element is the best action
# util.raiseNotDefined()
```

در این نوع الگوریتم، انتظار میرود روح ها همیشه بهترین عملکرد خود را اجرا نکنند بلکه در واقع با احتمال یکسان به یکی از جهت ها حرکت کنند.

پس از تابع value_expected به جای value_min استفاده شده است که مقدار و اکشن انتخابی، وابسته به مجموع ضرب احتمال در امتیاز به ازای هر گره فرزند است. یعنی به ازای هر اکشنی که میتواند رخ دهد، مقدار را در احتمال رخ داد ضرب میکنیم. نهایتا تمامی این مقادیر بدست آمده را با هم جمع کرده و به عنوان value_expected برمیگردانیم.

سوال: همانطورکه درسوال دوم اشاره شد روش مینیماکس درموقعیتی که در دام قرار گرفته باشد خودش اقدام به باختن و پایان سریع تر میکند ولی در صورت استفاده از مینیماکس احتمالی در 50 درصد از موارد برنده می شود. این سناریو را با هر دو روش امتحان کنید و درستی این گزاره را نشان دهید

```
PS D:\Python_project\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
```

```
PS D:\Python_project\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 emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Average Score: -88.4
Scores:        532.0, -502.0, -502.0, -502.0, -502.0, -502.0, 532.0, 532.0, -502.0, 532.0
Win Rate:      4/10 (0.40)
Record:        Win, Loss, Loss, Loss, Loss, Loss, Win, Win, Loss, Win
PS D:\Python_project\AI-P2>
```

تفاوتی که در این روش با minimax وجود دارد است این است که به جای اینکه پکمن روح ها را تماما عقلانی و پرفکت در نظر بگیرد.به این شکل در نظر میگیرد که روح ها در هر حرکت به طور شانسی به یک سمت می روند و عامل در زمانی که میداند اگر روح ها عقلانی عمل کنند میمیرد شانس خود را امتحان میکند تا شاید زنده بماند ولی در تفکر minimax پکمن فرض میکند روح ها بهترین اکشن خود را انجام میدهند برای همین بهترین عمل خود که خودکشی است را انتخاب میکند.

برای همین در روش expectimax پکمن توانسته است بعضی از بازی های خود را برنده شود.

تابع ارزیابی

سوال : تفاوت های تابع ارزیابی پیاده سازی شده در این بخش را با تابع ارزیابی بخش اول بیان کنید و دلیل عملکرد بهتر این تابع ارزیابی را بررسی کنید.

```
from util import manhattanDistance as mDist
newPos = currentGameState.getPacmanPosition()
newFoodPos = (currentGameState.getFood()).asList()
newGhostStates = currentGameState.getGhostStates()
newGhostPos = [ghostState.getPosition() for ghostState in newGhostStates]
newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

evaluatedValue = 0
for foodDist in newFoodPos:
    dist = mDist(newPos, foodDist)
    if dist < 5:
        evaluatedValue += 6 - dist
    else:
        evaluatedValue += 1

for ghostDist in newGhostPos:
    dist = mDist(newPos, ghostDist)
    if dist == 0:
        evaluatedValue = -1 * evaluatedValue
    elif dist < 5:
        evaluatedValue -= (6 - dist) * 1.5
    else:
        evaluatedValue += 1

return currentGameState.getScore() + evaluatedValue + sum(newScaredTimes)
```

به جای استفاده از successorsGameState از خود currentGameState استفاده شده کردم .اما اینکه چرا در اینجا به خوبی و حتی بهتر از سوال اول عمل میکند در ادامه بیان میکنم .همانطور که در اجرا تست کیس های مربوطه مشاهده میشود، پکمن به دنبال شکار کردن روح ها است !چون نسبتا وزن زیادی بر روی حالت scared روح ها داده ام، پک من وقتی که روح نسبتا نزدیکش است،به سمت کپسول حرکت کرده و در کنارش قرار میگیرد، سپس صبر میکند تا روح نزدیک تر شود و سپس روح را میخورد ! بدین صورت هم روح از او دور شده (روح از ابتدا شروع میکند) و هم خوردن روح امتیاز بسیار بیشتری از خوردن غذا به او میدهد .همچنین وقتی روح را شکار کرد و خیالش راحت شد، حال به خوردن غذا های اطرافش میپردازد .در نتیجه با این استراتژی میتواند امتیازهای بالایی بدست بیاورد.

بلاخره تموم شد😊