به نام خدا

گزارشکار پروژه دوم هوش مصنوعی

هلیاسادات هاشمی پور-۹۸۳۱۱۰۶

بخش اول: عامل عکس العمل

با توجه به دستور کار در این بخش قصد داریم که عملکرد ReflexAgent را بهبود ببخشیم. با توجه به کد ابتدا مقدار score را به صفر مقدار دهی اولیه کرده ایم. حال در حلقه تکراری که داریم فاصله ی منهتن موقعیت جدید پکمن و foodرا محاسبه کرده و اگر newPosکه همان محل جدید قرارگیری پکمن هست در آنجا food وجود داشته باشد score یک واحد زیاد شود. و در صورتی که مقدار آن مخالف صفر بود به صورت معکوس به scoreاضافه می کنیم. در حلقه ی تکرار دیگر این روند را برای موقعیت فعلی پکمن با ghost تکرار می کنیم در صورتی که بیشتر از یک باشد به score خود معکوس آن فاصله را اضافه می کنیم و اگر برابر یک بود مقدار ۱۰۰۰- (که بزرگترین مقدار منفی ما می باشد) را به عنوان امتیاز بر می گردانیم. در آخر هم مقدار امتیاز بازی را به score اضافه کرده و مقدار آن را بر می گردانیم.

```
def evaluationFunction(self, currentGameState, action):
    """
    Design a better evaluation function here.
    The evaluation function takes in the current and proposed successor
    GameStates (pacman.py) and returns a number, where higher numbers are better.
    The code below extracts some useful information from the state, like the
    remaining food (newFood) and Pacman position after moving (newPos).
    newScaredTimes holds the number of moves that each ghost will remain
    scared because of Pacman having eaten a power pellet.
    Print out these variables to see what you're getting, then combine them
    to create a masterful evaluation function.
    """
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
    "*** YOUR CODE HERE ***"
    # there is two situation for getting score
    score = 0
    foodDist = [util.manhattanDistance(newPos, foodPos) for foodPos in newFood.asList()]

    if len(foodDist) > 0:
        score += float(1 / min(foodDist))
    else:
        score += 1

    for ghost in newGhostStates:
        distance = abs(newPos[0] - ghost.getPosition()[0]) + abs(newPos[1] - ghost.getPosition()[1])
        if distance > 1:  # there is not ghost & check the new position
            score += float(1 / distance)
        # return large negative value
        elif distance == 1:
            return -1000

    score += successorGameState.getScore()
    return score
```

با استفاده از دستور زیر نتایج زیر حاصل می شود.

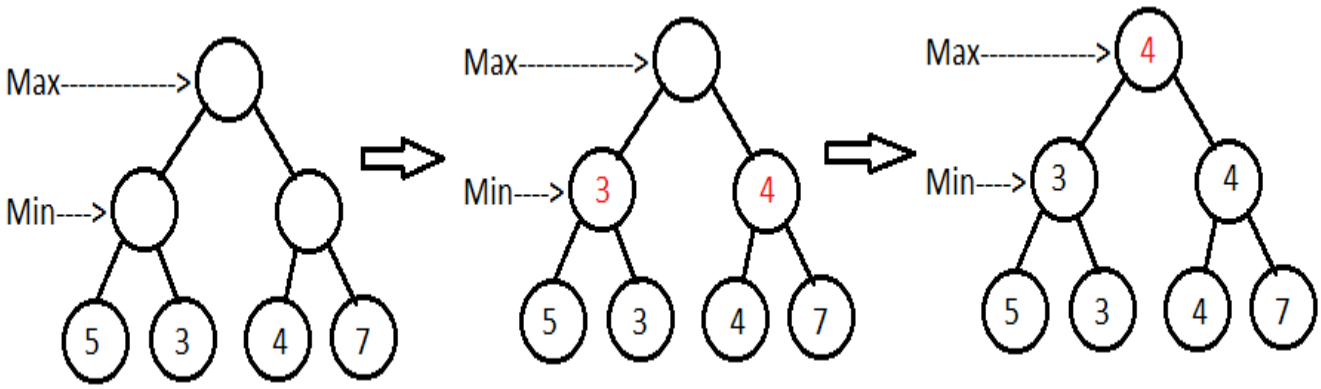
```
python autograder.py -q q1 --no-graphics
```

```
Question q1
===========

Pacman emerges victorious! Score: 1219
Pacman emerges victorious! Score: 1228
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1227
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1229
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1228
Pacman emerges victorious! Score: 1224
Pacman emerges victorious! Score: 1211
Average Score: 1225.9
```

بخش دو: مینیماکس

در این بخش می خواهیم Minimax را پیاده سازی کنیم. می دانیم دو نوع گره در درخت الگوریتم مینیماکس وجود دارد.(Max nodeو Min node).ما باید بیشینه ی مقدار را در که حداقل مقداری که می توانیم به دست آوریم، پیدا کنیم. ابتدا دو تابع به نام های maximumValue و minimumVale داریم. طبق دستور کار عامل ما باید به ازای هر تعداد روح درست کار بکند.درخت مینیماکس ما باید تا عمق دلخواه گسترش یابد. شکل زیر، به مفهوم مینیماکس اشاره می کند.



در تابع maximumValue ابتدا مقدار maxVale را به منفی بینهایت مقداردهی کردیم و هر بار از عمق درخت ما یکی کم می شود. شرط های لازم را هم بررسی می کنیم که آیا به استیت نهایی یا گره ی برگ رسیده ایم یا خیر.(در دستور کار هم ذکر شده است که scoreEvaluationFunction براساس استیت ارزیابی را انجام می دهد.)در حلقه ی تکرار ، بیشینه مقدار را بررسی کرده(در اصل مقدار maxVale را به روز رسانی می کنیم) و در نهایت آن را بر می گرداند.(با توجه به شکل بالا می توان دلیل به روند نوشتن کد پی برد در اصل بین مقدار maxValue در حلقه ی تکرار و مقدار کمینه ای که داشتیم مقایسه صورت گرفته و در نهایت مقدار بیشینه در این مرحله انتخاب می شود.)

```
def maximumValue(gameState, depth):
    maxValue = float('-inf')
    depth = depth - 1  # reduce the size of depth
    # Returns a list of legal actions for an agent & ghosts are >= 1
    legalActions = gameState.getLegalActions(0)

    # conditions for end
    if depth == 0:
        return self.evaluationFunction(gameState)
    if gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)  # Returns the minimax action from the current gameState

    # get the max value from all its successor
    for action in legalActions:
        # Returns the successor game state after an agent takes an action
        maxValue = max(maxValue, minimumValue(gameState.generateSuccessor(0, action), depth, 1))
    return maxValue
```

در تابع minimumValue ابتدا مقدار minVale را به مثبت بینهایت مقداردهی کردیم. legalActionدر این تابع به کارهای مجازی که عامل روح ما می تواند انجام بدهد اشاره می کند.(موقعیت های مجاز البته برای درک بهتر می توان آن را همان شاخه های درخت دانست)سپس با توجه به کد نوشته شده بر اساس مقدار agentIndex و تعداد روح ها مقدار کمینه را بر می گردانیم.(برای مثال در شرط چک می شود که(ghostNumbers-1) آیا روح باقی مانده است یا خیر و با توجه به آن در شرط نخست(برای روح های باقی مانده است یعنی آخرین روح نباشد به پیمایش ادامه داده و agentIndex را به علاوه ی یک می کنیم) کمینه ی مقدار را بر میگرداند و در شرط بعدی بیشینه را بر می گرداند(در اصل آخرین روح بوده و به روند عادی ادامه می دهد).

```
def minimumValue(gameState, depth, agentIndex):
    minValue = float('inf')
    legalActions = gameState.getLegalActions(agentIndex)

    if gameState.isWin() or gameState.isLose():  # the conditions for end
        return self.evaluationFunction(gameState)  # Returns the minimax action from the current gameState

    for action in legalActions:
        if agentIndex < (ghostsNumbers - 1):
            minValue = min(minValue,
                           minimumValue(gameState.generateSuccessor(agentIndex, action), depth, agentIndex + 1))
        else:
            minValue = min(minValue, maximumValue(gameState.generateSuccessor(agentIndex, action), depth))
    return minValue
```

در تابع result بیشینه مقدار از successorهایش را می گیرد و در شرطی که هست بزرگترین مقدار را بر می گرداند. در اقع ابتدا max node را با maxValue مقداردهی می کنیم.(منفی بی نهایت است) در حلقه ی تکرار تمامی راه های ممکن بررسی می شود و مقداری که باید حل شود باید مقداری که بر روی mini node است باشد و با max nodeای که مقداردهی شده است مقایسه شود. در حلقه ی تکرار و بخش شرط آن مشاهده می کنیم اگر actionValue بزرگتر از maxValueباشد مقدار maxValue به روز رسانی شده (در اصل مقدار مینیماکس برای max-playerرا محاسبه می کند) درآخر مقدار نهایی بر گردانده می شود. در واقع بهترین انتخاب بعد از پیمایش برگردانده می شود.

```
def result():
    maxValue = float('-inf')
    res = 0

    for action in gameState.getLegalActions(0):
        Action = self.depth
        actionValue = minimumValue(gameState.generateSuccessor(0, action), Action, 1)  # Returns the minimax action from the current g
        if actionValue > maxValue:
            maxValue = actionValue
            res = action
    return res

return result()
```

با استفاده از دستور زیر نتایج زیر حاصل می شود.

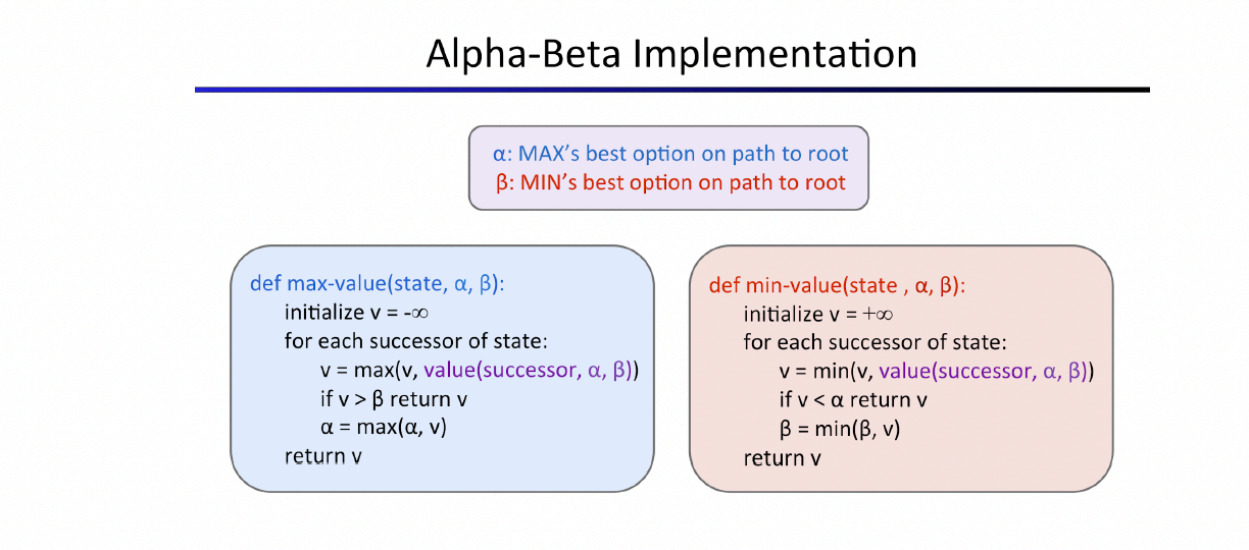
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

```
(venv) heliaa@MacBook-Pro multiagents % python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501.0
Win Rate:      0/1 (0.00)
Record:        Loss
```

در این حالت پکمن به این نتیجه می رسد که مردن آن اجتناب ناپذیر است و برای اینکه امتیازش کم نشود،تلاش می کند زودتر ببازد.

بخش سه: هرس آلفا-بتا

این بخش همچون قسمت پیشین هست فقط خاصیت هرس آلفا و بتا را به درخت مینیماکس اضافه کرده ایم. در اصل الگوریتم آلفا و بتا بهینه الگوریتم مینیماکس است. که مینیماکس یک الگوریتم جامع می باشد که باید از تمامی گره ها عبور کند. الگوریتم آلفا بتا می تواند کارایی مینیماکس را با هرس کردن و کم کردن گره هایی که ضرروری نیستند بهبود بخشد.آلفا نشان دهنده ی حداکثر کران پایین و بتا نشان دهنده ی حداکثر کران بالا برای بهترین گزینه در مسیر ریشه است. حال اگر آلفا از بتا بزرگتر باشد، نشان دهنده ی این است که این نقطه راه حل بهینه ای برای ما نمی باشد. با توجه به شکل پایین که در دستور کار هم آمده است بخش های خواسته شده را به کد اضافه کرده ام.



تابع maximumValue

همانطور که ذکر کردم توضیحات این تابع همانند قسمت پیشین هست و فقط قسمتی که کادر قرمز دور آن کشیده شده است با توجه به کدی که در دستور کار هم مشخص شده است ، اضافه شده است.

```
def maximumValue(gameState, depth, Alpha, Beta):
    legalActions = gameState.getLegalActions(0)
    depth = depth - 1
    actionValue = float('-inf')  # max value

    # conditions for end
    if depth == 0:
        return self.evaluationFunction(gameState)

    if gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)

    for action in legalActions:
        actionValue = max(actionValue,
                          minimumValue(gameState.generateSuccessor(0, action), depth, 1, Alpha, Beta))
        if actionValue > Beta:
            return actionValue
        Alpha = max(Alpha, actionValue)
    return actionValue
```

تابع minimumValue

همانطور که ذکر کردم توضیحات این تابع همانند قسمت پیشین هست و فقط قسمتی که کادر قرمز دور آن کشیده شده است با توجه به کدی که در دستور کار هم مشخص شده است ، اضافه شده است.

```
def minimumValue(gameState, depth, agentIndex, Alpha, Beta):
    actionValue = float('inf')  # minValue
    legalActions = gameState.getLegalActions(agentIndex)

    if gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)

    for action in legalActions:
        if agentIndex < (ghostsNumbers - 1):
            actionValue = min(actionValue, minimumValue(gameState.generateSuccessor(agentIndex, action), depth,
                                                        agentIndex + 1, Alpha, Beta))
        else:
            actionValue = min(actionValue,
                              maximumValue(gameState.generateSuccessor(agentIndex, action), depth, Alpha, Beta))
        if actionValue < Alpha:
            return actionValue
        Beta = min(Beta, actionValue)
    return actionValue
```

تابع result

این تابع هم همچون قسمت پیشین پیاده سازی شده است اما با توجه به نیاز این قسمت، در قسمتی که کادر قرمز کشیده شده است شرط را عوض کرده ام.

```
def result():
    res = 0
    maxValue = float('-inf')
    Alpha = float('-inf')
    Beta = float('inf')
    legalActions = gameState.getLegalActions(0)

    for action in legalActions:
        Action = self.depth
        actionValue = minimumValue(gameState.generateSuccessor(0, action), Action, 1, Alpha, Beta)
        if actionValue > Alpha:
            res = action
            Alpha = actionValue
    return res

return result()
```

بخش چهار: مینیماکس احتمالی

این بخش هم مانند بخش دو می باشد اما در کد تغییراتی ایجاد شده است در اصل از مینیماکس احتمالی استفاده کرده ایم که در آن به جای در نظر گرفتن کوچک ترین حرکات حریف، مدلی از احتمال حرکات حریف را در نظر می گیرد.در تابع maximumValue تغییراتی ایجاد نشده است اما در تابع minimumValue تغییرات به صورت زیر است. در اصل ما برای هر روح،Expection valueرا محاسبه کرده و جمع expValueها را در متغیرsum انجام داده و در هر پیمایش میانگین مقادیر را به دست آورده و در انتها آن را بر می گردانیم.

```
def getExpectation(gameState, depth, agentIndex):
    sum = 0
    result = 0
    legalActions = gameState.getLegalActions(agentIndex)
    num_actions = len(gameState.getLegalActions(agentIndex))

    if gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)

    for action in legalActions:
        if agentIndex < (ghostsNumbers - 1):
            expValue = getExpectation(gameState.generateSuccessor(agentIndex, action), depth, agentIndex + 1)
        else:
            expValue = maximumValue(gameState.generateSuccessor(agentIndex, action), depth)
        sum += expValue
        result = float((sum) / (num_actions))
    return result
```

با استفاده از دستور زیر نتایج زیر حاصل می شود.

python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10

```
(venv) heliaa@MacBook-Pro multiagents % 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
```

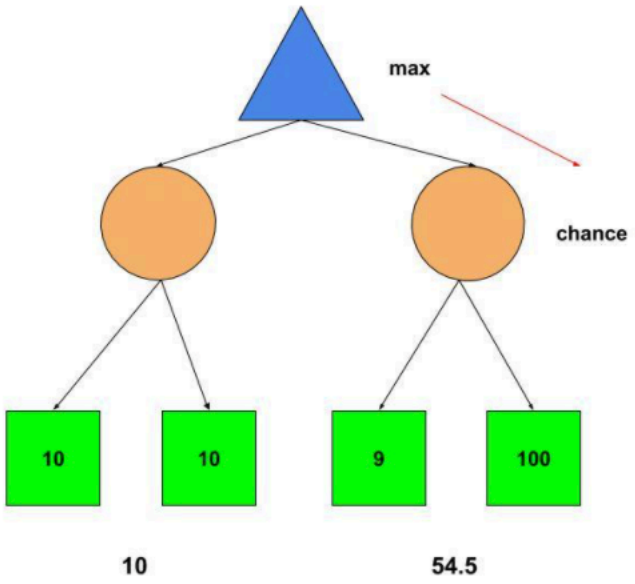
با استفاده از دستور زیر نتایج زیر حاصل می شود.

python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10

```
(venv) heliaa@MacBook-Pro multiagents % python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Average Score: 221.8
Scores:        532.0, 532.0, 532.0, 532.0, 532.0, -502.0, -502.0, 532.0, 532.0, -502.0
Win Rate:      7/10 (0.70)
Record:        Win, Win, Win, Win, Win, Loss, Loss, Win, Win, Loss
```

برای درک قسمت نوشته شده در کد می توان به مثال زیر توجه کرد که یک میانگینی ازutility ها می گیریم.
(10+10)/2=10
(100+9)/2=54.5

حال maximizer زیر درخت راست را انتخاب می کند.



بخش پنجم: تابع ارزیابی

در این بخش باید این تابعی که نوشتیم،حالت ها را ارزیابی کند و باید تابعی بنویسیم که امتیاز پکمن را افزایش دهد.ابتدا همچون بخش اول عمل کردیم، اگر newPosکه همان محل جدید قرارگیری پکمن هست در آنجا food وجود داشته باشد score یک واحد زیاد شود. حال در حلقه تکراری که داریم فاصله ی منهتن موقعیت جدید پکمن و food را محاسبه کرده و در صورتی که مقدار آن مخالف صفر بود به صورت معکوس به score اضافه می کنیم. سپس فاصله را تا ghsot بررسی می کنیم. معکوس فاصله ی منهتنی را از scoreکم می کنیم حال اگر فاصله ی ghost تا پکمن ما کم تر مساوی یک باشد به مقدار ghostsNear یکی اضافه کرده تا در انتهای کار از امتیاز کل کم کنیم و اگر مساوی یک بود بزرگترین مقدار منفی را بر می گردانیم.(البته این بخش نیازی نبود چون توسط شرط قبلی پوشش داده شده بود)

ما در بازی کپسول هایی داریم بنابراین در این بخش آن کپسول ها را هم در نظر می گیریم و در نهایت از scoreآن را کم می کنیم. برای اینکه به عملکرد بهتری برسیم من ضریب هایی را در نظر گرفته ام.مثلا کپسول ها را در عدد منفی بزرگی مثل منفی ۲۰ ضرب کردم تا برای پکمن انگیزه در ایجاد شود تا کپسول هایی که از آن ها می گذرد را بخورد. (امتیاز هر کپسول ۲۰+ است) .

```
def betterEvaluationFunction(currentGameState):
    """
    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 ***"
    newPos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    score = 0
    newGhostStates = currentGameState.getGhostPositions()

    # Feature 1: food positions
    foodDist = [util.manhattanDistance(newPos, foodPos) for foodPos in newFood.asList()]

    if len(foodDist) > 0:
        score += float(1 / min(foodDist))
    else:
        score += 1

    ghostsDis = 1
    ghostsNear = 0
    # Feature 2:  distances from ghosts if exists
    for ghost in newGhostStates:
        distance = util.manhattanDistance(newPos, ghost)
        ghostsDis += util.manhattanDistance(newPos, ghost)
        score -= float(1 / (ghostsDis))
        if distance <= 1:
            ghostsNear += 1
        elif distance == 1:
            return -10000
    # Feature 3: capsules positions
    numberOfCapsules = len(currentGameState.getCapsules())

    score = score + currentGameState.getScore() - 2 * ghostsNear - 20 * numberOfCapsules
    return score
```

با استفاده از دستور زیر نتایج زیر حاصل می شود.

python autograder.py -q q5 --no-graphics

```
Question q5
===========

Pacman emerges victorious! Score: 940
Pacman emerges victorious! Score: 1112
Pacman emerges victorious! Score: 1173
Pacman emerges victorious! Score: 1370
Pacman emerges victorious! Score: 1174
Pacman emerges victorious! Score: 1172
Pacman emerges victorious! Score: 1074
Pacman emerges victorious! Score: 1126
Pacman emerges victorious! Score: 1247
Pacman emerges victorious! Score: 1169
Average Score: 1155.7
```