باسمه تعالی

گزارش پروژه دو درس هوش مصنوعی – دکتر جوانمردی

سید امیرمهدی میرشریفی – ۹۸۳۱۱۰۵

۱) عامل عکس العمل

سوال: توضیح دهید که از هر کدام از پارامترهای دخیل در تابع ارزیابی چگونه استفاده کرده‌اید و هر کدام چگونه برروی خروجی تاثیر می گذراند (تاثیر کدام یک از پارامترها بیشتر است؟).

سوال: چگونه می‌توان پارامترهایی که مقادیرشان در یک راستا نمی‌باشند را با یکدیگر برای تابع ارزیابی ترکیب کرد؟ (مانند فاصله تا غذا و روح که ارزش آن‌ها بر خلاف یکدیگر می‌باشد)

پاسخ:

ابتدا به سراغ پاسخ سوال دوم می رویم .در این سوال از ۴ پارامتر که عبارتند از : فاصله تا غذا، فاصله تا روح، موقعیت پکمن و شمارشگر ترس روح استفاده می کنیم. برای تخمین امتیاز این مرحله یک متغیر به نام point تعریف میکنیم. در واقع این متغیر حاوی مقدار ضرری است که به واسطه نزدیکی روح ها و فاصله تا غذا کسب می کنیم. برای تکمیل متغیر point به این صورت عمل می کنیم که نزدیک ترین فاصله تا غذا را به دست می آوریم و بعلاوه point میکنیم. تا اینجا هزینه خوردن غذا به دست می آید. سپس می بایست تاثیر روح ها را نیز به دست بیاوریم به این صورت که اگر زمانی که روح به موقعیت پکمن میرسد همچنان ترسیده باشد که به حساب نمی آید و اگر خیر می بایست میزان تاثیر آن را (متناسب با فاصله روح تا پکمن) اضافه کرد به متغیر point. با این روش تاثیر خوردن غذا خلاصه در هزینه تا رسیدن به آن و همسو با تاثیر ارواح بازی می شود.
حال به توضیح جزئی تر کد که پاسخ سوال اول هم نیز هست می رویم:

```
successorgameState = currentgameState.generatePacmanSuccessor(action)
newPos = successorgameState.getPacmanPosition()
newFood = successorgameState.getFood()
newGhostStates = successorgameState.getGhostStates()
foodNum = currentgameState.getFood().count()
newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
point=0
```

این بخش از کد تعریف های اولیه از وضعیت جدید پکمن بعد از اعمال حرکت است که از آن موقعیت جدید و اطلاعات غذا های باقی مانده و ارواح بازی و همچنین شمارشگر ترس آن ها و متغیر point را تعریف میکنیم.

```
point=0
if len(newFood.asList()) == foodNum:
       point = 10000000
for food in newFood.asList():
        point=min(manhattanDistance(food , newPos) , point)
```

در این بخش متغیر point را ابتدا صفر تعریف میکنیم. سپس اگر تعداد غذا در این مرحله و مرحله قبل یکی باشد یعنی پکمن هیچ غذایی نخورده است. بنابراین برای متغیر point یک مقدار بسیار بزرگ تعریف میکنیم که در حلقه بعدی و هنگام به دست آوردن کم ترین هزینه و موقع اعمال تابع مینیمم ، هزینه مسیر انتخاب میشود. اما اگر پکمن غذایی خورده باشد مقدار همان صفر می ماند و این یعنی در حلقه بعد هیچ هزینه ای برای رسیدن به غذا به حساب نمی آید.

```
 for ghost in newGhostStates:
     if manhattanDistance(ghost.configuration.pos,newPos) <= point :
        if manhattanDistance(ghost.configuration.pos,newPos)>=ghost.scaredTimer:
           point += 10 ** (2 -manhattanDistance(ghost.configuration.pos, newPos))
return -point
```

در این مرحله میخواهیم تاثیر روح را بررسی کنیم. اگر فاصله روح تا پکمن از مقدار شمارشگر ترس آن کمتر باشد یعنی وقتی روح به موقعیت پکمن می رسد فعال است و میشود تاثیر آن را محاسبه کرد.

از آن جایی که تاثیر ارواح تاثیر قوی تری هستند از توان استفاده میکنیم که پایه آن را میتوان هر عدد بزرگتر از ۴ ای انتخاب کرد(به صورت تجربی) و توان آن هم برابر ۲ (اگر پکمن و روح در یک جهت و به سمت هم حرکت کنند در یک حرکت دو خانه را طی خواهند کرد) – منهای فاصله بین روح و پکمن است.

در آخر از آنجایی که مقدار point از جنس هزینه است مقدار منفی آن را بر می گردانیم.

همانطور که مشاهده می کنید تمام پارامتر ها استفاده شده اند.

*** تابع value()

در این بخش تابعی مهم را معرفی خواهیم کرد که برگرفته از اسلاید های درسی است و از آن در سه بخش بعدی یعنی minimax, alphaBetaAgent , ExpectimaxAgent استفاده خواهیم کرد.

این تابع را جزو متد های کلاس MultiAgentSearchAgent تعریف میکنیم.

ورودی این تابع شماره اندیس عامل ، عمق درخواستی، وضعیت بازی، آلفا و بتا و این که احتمالی هست یا خیر (الفا و بتا و احتمالی بودن اختیاری است)

```
def value(self,agent, depth, gameState,alpha=None,beta=None,isExpectimax=False):
if alpha is None  and  beta is None:
```

```
def value(self,agent, depth, gameState,alpha=None,beta=None,isExpectimax=False):
    if alpha is None  and  beta is None:
        if gameState.isLose() or gameState.isWin() or depth == self.depth:
            return scoreEvaluationFunction(gameState)
        if agent == 0:
            return max(self.value(1, depth, gameState.generateSuccessor(agent, newState)) for newState in gameState.getLegalActions(age
        else:
            nextAgent = (agent + 1) %(gameState.getNumAgents())
            if nextAgent == 0:depth+=1
            if isExpectimax: return sum(self.value(nextAgent, depth, gameState.generateSuccessor(agent, newState),isExpectimax=True) fo
            return min(self.value(nextAgent, depth, gameState.generateSuccessor(agent, newState)) for newState in gameState.getLegalAct
```

در این بخش از کد که مختص بخش سوال minimax یا Expectimax است مشخص میشود اگر برای تابع، آلفا و بتا که نشان دهنده بخش هرس هستند ، تعریف نشده باشد می بایست این بخش اجرا شود.
در ابتدا چک میشود اگر در وضعیت برد یا باخت یا به انتهای عمق مدنظر رسیده باشیم تابع تخمین صدا زده شود. درغیر این صورت اگر عامل پکمن باشد که شماره ایندکس آن صفر است ماکسیمم مقدار بین مقادیر بازگشتی همین تابع برای ارواح انتخاب شود. اگر عامل روح باشد ابتدا یکی به آن اضافه میکنیم . اگر پکمن شد مجدد همین تابع را به صورت بازگشتی صدا میزنیم اما با عمق یکی بیشتر. اما اگر روح بعدی بود با همین مشخصات صدا زده میشود با فرق شماره ایندکس عامل. اگر توجه کنید در هنگام انتخاب مقدار برای ارواح بررسی میکنیم که اگر از ما احتمالات خواسته شده بود بجای حداقل مقدار مجموع آن را برمیگردانیم. تا در این مسئله برای تمام مقادیر ۳ برابر امید ریاضی باشد(فرقی نمیکند که امید ریاضی برگردانده شود یا جمع)
در بخش بعدی که کد آن در ادامه آمده است بخش مختص هرس کردن است که شبیه کدقبلی است اما در هر مرحله آلفا و بتای مربوطه نیز پاس داده میشود و عملیات های مربوط به هرس کردن اعمال میشود.

```
else:
     if gameState.isLose() or gameState.isWin() or depth == self.depth:
       return scoreEvaluationFunction(gameState)
     if agent==0:
         maxval=-100000
         for action in gameState.getLegalActions(agent):
         maxval=max(maxval,self.value(1, depth,
gameState.generateSuccessor(agent, action),alpha,beta))
         if maxval > beta : return maxval
             alpha=max(maxval,alpha)
             return maxval
         else:
                minval=100000
                nextAgent = (agent + 1) %(gameState.getNumAgents())
```

```
                if nextAgent == 0:depth+=1
                for newState in gameState.getLegalActions(agent):
                    minval=min(minval,self.value(nextAgent, depth,
gameState.generateSuccessor(agent, newState),alpha,beta))
                    if minval<alpha: return minval
                    beta=min(beta,minval)
```

۲) مینیماکس

همانطور که در بالا و در معرفی تابع value گفته شد دیگر الان صرفا نیاز به این است که پکمن به ازای تمام اکشن ها و روح اول این تابع را فراخوانی و مقدار و حرکت بهینه را به دست بیاورد.

```
maximum_val = -1000000
optimal_action = Directions.STOP
for agentState in gameState.getLegalActions(0):
    value = self.value(1, 0, gameState.generateSuccessor(0,
agentState),alpha,beta)
      if value > maximum_val :
                maximum_val = value
                optimal_action = agentState

 return optimal_action
```

سوال: وقتی پکمن به این نتیجه برسد که مردن آن اجتناب ناپذیر است، تلاش می‌کند تا به منظور جلوگیری از کم شدن امتیاز، زودتر ببازد. این موضوع را می‌توانید با اجرای دستور زیر مشاهده کنید:

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

بررسی کنید چرا پکمن در این حالت به دنبال باخت سریع‌تر است؟

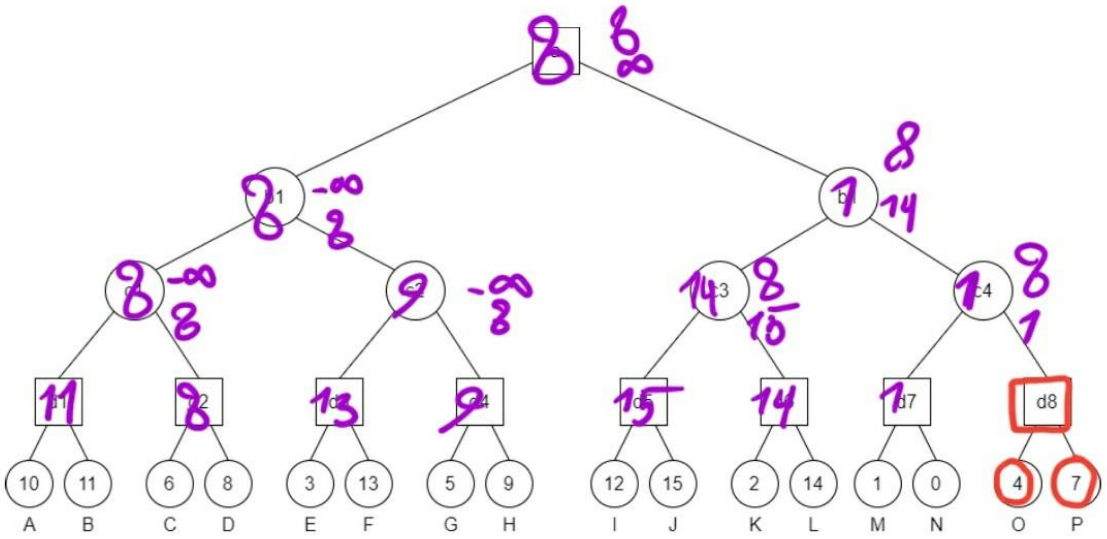
پاسخ: به این دلیل که در این حالت با فرض این که حریف همیشه زیان بار ترین حرکات را علیه ما به کار میگرد و اگر راهی برای بردن انتخاب نکند و با گذشت زمان ضرر ما بیشتر میشود ، سودمندی را در ضرر کمتر و مرگ زودتر پیدا میکند.

۳) هرس آلفا بتا

در این بخش هم دقیقا همانند بخش قبل فقط با تفاوت این که می بایست آلفا و بتا را تعریف کنیم.

```
maxval = -100000
action = Directions.STOP
alpha = -100000
beta = 100000
for agentState in gameState.getLegalActions(0):
    ghostValue = self.value(1, 0, gameState.generateSuccessor(0, agentState), alpha, beta)
     if ghostValue > maxval:
                maxval = ghostValue
                action = agentState
      if maxval > beta:
                return maxval
           alpha = max(alpha, maxval)
       return action
```

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



پاسخ: همانطور که مشاهده می کنید تمام مراحل با مقادیر آلفا و بتا مشخص شده است و گره های قرمز هرس شده اند. در وضعیت فعلی سودمندی پکمن در این است که به سمت چپ درخت حرکت کند.

۴) مینیماکس احتمالی

این بخش هم همانند بخش های قبل با استفاده از تابع value، ضمن true کردن فیلد isExpectimax

```
class ExpectimaxAgent(MultiAgentSearchAgent):
    def getAction(self, gameState):
        maximum_val = -100000
        optimal_action = Directions.STOP
        for agentState in gameState.getLegalActions(0):
            value = self.value(1, 0, gameState.generateSuccessor(0, agentState),isExpectimax=True)
            if value > maximum_val :
                maximum_val = value
                optimal_action = agentState
        return optimal_action
```

سوال: همانطور که در سوال دوم اشاره شد روش مینیماکس در موقعیتی که در دام قرار گرفته باشد خودش اقدام به باختن و پایان سریع‌تر بازی می‌کند ولی در صورت استفاده از مینیماکس احتمالی در ۵۰ درصد از موارد برنده می‌شود. این سناریو را با هر دو دستور زیر امتحان کنید و درستی این گزاره را نشان دهید. همچنین دلیل این تفاوت در عملکرد مینیماکس و مینیماکس احتمالی را توضیح دهید.

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

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

پاسخ:

در ادامه نرخ برد و رکورد های ثبت شده در دو حالت نشان داده شده است که نتیجه اجرا گرفتن دو کد بالا برای هر بخش است:

مینیماکس:

Win Rate: 0/10
Record: Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

مینیماکس احتمالی:

Win Rate: 3/10
Record: Loss, Win, Loss, Loss, Loss, Win, Loss, Loss, Loss, Win

*نکته: در صورت سوال درصد احتمال را ۵۰ بیان کرده است که با توجه به احتمالی بودن این درصد هم برای چند اجرا احتمالی است. برای مثال در اجرا های بعدی کد در بخش مینیماکس احتمالی این نرخ برد هشتاد درصد یا چهل درصد هم ثبت شد.

و اما دلیل این اتفاق این است که در بخش مینیماکس احتمالی ما از امید ریاضی برای هر گره استفاده می کنیم که در این حالت نتیجه برای انتخاب حداقل و حداکثر ها فرق خواهد کرد . بنابراین پکمن هم با توجه به این قضیه انتخاب هایش متفاوت خواهد شد.

سوال: الگوریتم رولت ویل را بررسی کنید و بیان کنید که انتخاب هر کروموزوم در این الگوریتم بر چه اساسی است؟ اگر در بازی پکمن خودمان از آن استفاده کنیم، چه معیاری برای انتخاب هر **action** مناسب است؟ بر فرض اگر نیاز بود تا با کمک الگوریتم رولت ویل بیش‌تر از یک حالت انتخاب شود (با کمک مقدار تابع ارزیابی برای هر حالت) و درخت با توجه به این دو حالت گسترش پیدا کند و حالت‌های بعدی آن‌ها هم بررسی شوند (تا بتوانیم برای حالت بعدی انتخاب بهتری داشته باشیم)، چه راهی به نظر شما منطقی می‌باشد؟
پاسخ:

در این الگوریتم ما ارزش و خوبی هر کروموزوم را بر اساس یک تابع ارزش به دست می آوریم و سپس احتمال متناظر با آن برابر مقدار تابع ارزش آن تقسیم بر مجموع کل این مقدار برای تمام کروموزوم ها ست. به عبارتی دیگر احتمال انتخاب در این مسئله به میزان خوب بودن کروموزم بستگی دارد. هرچه بهتر احتمال انتخابش بیشتر.

در بازی ما میشود بجای تابع ارزش برای کروموزوم از تابع تخمین برای یک حرکت که در بخش اول طراحی کردیم استفاده کرد و در مواقعی که احتمالات داریم خوبی و ارزش یک اکشن را نسبت به کل تعیین و آن را احتمال وقوع آن اکشن بگذاریم. اگر بخواهیم چند اکشن را برگزینیم میتوانیم در هر سطح به تعداددلخواه بهترین ها را انتخاب کنیم و صرفا فرزند های آنان را گسترش دهیم.

۵) تابع ارزیابی

در این بخش همان تابع ارزیابی بخش اول به کار رفته با تفاوت نبودن اکشن و حالت دومی برای بازی

*در تمام سوالات به جز بخش ۵ نمره کامل کسب شد. در بخش ۵ ، ۳ از ۶ کسب شد. لطفا در صورت مغایرت به بنده اطلاع بدهید.سپاس