دانشگاه صنعتي اميركبير
(پلی‌تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

پروژه سوم

نگارش
ایلیا راوند

استاد درس
پروفسور مهدی قطعی

استاد دوم
بهنام یوسفی مهر

فهرست مطالب

چکیده

در این ارائه قصد داریم درمورد الگوریتم های هوش مصنوعی در بازی پک من صحبت کنیم و آن هارا بررسی کنیم و آن هارا پیاده سازی کنیم.

الگوریتم های مورد بررسی ما این ها هستند:

1-الگوریتم مینیماکس(minimax)
2-الگوریتم الفا-بتا (alpha-beta)
3-الگوریتم اکسپکتی مکس(expectimax)
در جلو تر تمام الگوریتم های مورد نظر را شرح میدهیم و علاوه بر آن ها راجع به عامل انعکاسی (ReflexAgent)
صحبت میکنیم وآن را پیاده سازی میکنیم.

کلیدواژه

1-minimax
2-alpha-beta
3-expectimax
4-reflex agent

فصل اول : مقدمه

ماهیت این ارائه درباره این است ما اگر قرار باشد دو و یا چند عامل هوشمند داشته باشیم که قرار باشد با هم رقابت کنند چگونه باید حرکت های هم را پیشبینی و بهترین حرکت را کنند.
همانطور که میدانیم وقتی درباره حرکت های موجود در حالت خاص صحبت میکنیم پای درخت محاسبه وسط می‌اید و ما باید چگونه محاسبه کردن و ارزش گذاری مناسب برای هر نود این درخت را بدست بی‌آوریم.
برای گذاشتن این ارزش گذاری ها ما نیاز داریم به اینکه بتوانیم "وضعیت" را برای هر نود تعریف کنیم.
اما منظور ما از "وضعیت" چیست؟
بیایید با یک مثال توضیح دهیم. چگونه میشود وقتی دو نفر در حال بازی شطرنج هستند نفر سوم به صورت لحظه ای صفحه شطرنج را میبیند و میگوید که بازیکن سفید در حال باخت بازی است؟
مثلا میتواند از چینش مهره های مختلف روی صفحه شطرنج این حرف را بزند و یا از روی مهره های موجود بر روی صفحه این حرف را بزند.
اما کامپیوتر نمیتواند صحبت های من و شما را بفهمد چرا که فقط اعداد را میفهمد. پس ما باید بتوانیم که ارزش های موجود خودمان را به اعداد تبدیل کنیم.
در وهله اول باید بتوانیم ارزش گذاری کنیم و بعد از آن درخت آن را پیدا کنیم.
موضوع بعدی که برای ما پیش می‌آید این اسن گاهی ممکن است این درخت خیلی بزرگ شود و زمان و حافظه زیادی از ما بخواهد پس باید بتوانیم آن را کنترل کنیم.
خب حالا فرض کنید ما توانستیم که ارزش گذاری هارا انجام میدهیم اما وقتی دو طرف دعوا وجود دارد در بازی همیشه یک تیم در حال تخریب و آن یکی تیم در حال ساخت هست و اینجا الگوریتم مینی ماکس است که این کار را برای ما در درخت انجام میدهد و محاسبه میکند و بهترین حرکت را پیشنهاد میدهد که در فصل های بعدی آن را بررسی میکنیم.

نحوه انتخاب عامل:

الگوریتم مینیماکس و هرس الفا بتا هر دو یکسان عمل میکنند اما هرس الفا بتا به زمان کمتری برای اجرا نیازمند است و به همین دلیل قادر به تحلیل درخت حالات تا عمق بیشتری نسبت به الگوریتم مینیماکس است

پس همواره انتخاب الگوریتم هرس الفا بتا سودمند تر است.

الگوریتم اکسپکتی ماکس از لحاظ زمان مورد نیاز برای اجرا با الگوریتم مینیماکس برابری میکند و از این لحاظ نسبت به هرس الفا بتا بدتر است.

اما در شرایطی که حریف همواره بهینه عمل نمیکند و به صورت شانسی تصمیم میگیرد یا این که ممکن است مرتکب خطا شود، الگوریتم اکسپکتی ماکس بهتر از دو الگوریتم دیگر عمل میکند چرا که شرایطی که امکان دارد حریف خطا انجام دهد و عامل بتواند امتیاز بیشتری نیز کسب کند را نیز در نظر میگیرد.

فصل دوم : عامل انعکاسی(Reflex Agent)

عامل انعکاسی عاملی است که با توجه به شرایط در دسترس تصمیم میگیرد که چه کاری را انجام دهد و این کار را بدون در نظر گرفتن عواقب آن انتخاب آن را انجام میدهد.
اما این کار در مثال ما چگونه است و شرایط موجود را چگونه تحلیل میکند؟
کار اصلی ما عدد گذاری روی حالت های بازی هست و یک امتیاز نسبت میدهیم و قرار است پک من حالتی که بیشترین امتیاز را دارد انتخاب کند .
در ادامه پیاده سازی آن را می‌آوریم و خط به خط آن را شرح میدهیم.

پیاده سازی:

```
77          "*** YOUR CODE HERE ***"
78          #score = successorGameState.getScore()
79          score = 0
80
81          #so that pacmans dont get stop in some corner
82          if action == 'Stop':
83              score -= 100
84
85          #distance to the closest food
86          foodList = newFood.asList()
87          minDistance = 100000
88          score += -len(foodList) * 100
89          for food in foodList:
90              distance = manhattanDistance(food, newPos)
91              if distance < minDistance:
92                  minDistance = distance
93          #making pacman to go to the closest food
94          score += -minDistance
95          if foodList == []:
96              score += 10000 + minDistance
97
98          #if neraest ghost is close to pacman, pacman should avoid it
99          for ghost in newGhostStates:
100             ghostPos = ghost.getPosition()
101             distance = manhattanDistance(ghostPos, newPos)
102             if distance < 2:
103                 score -= 100000
104
105         return score
```

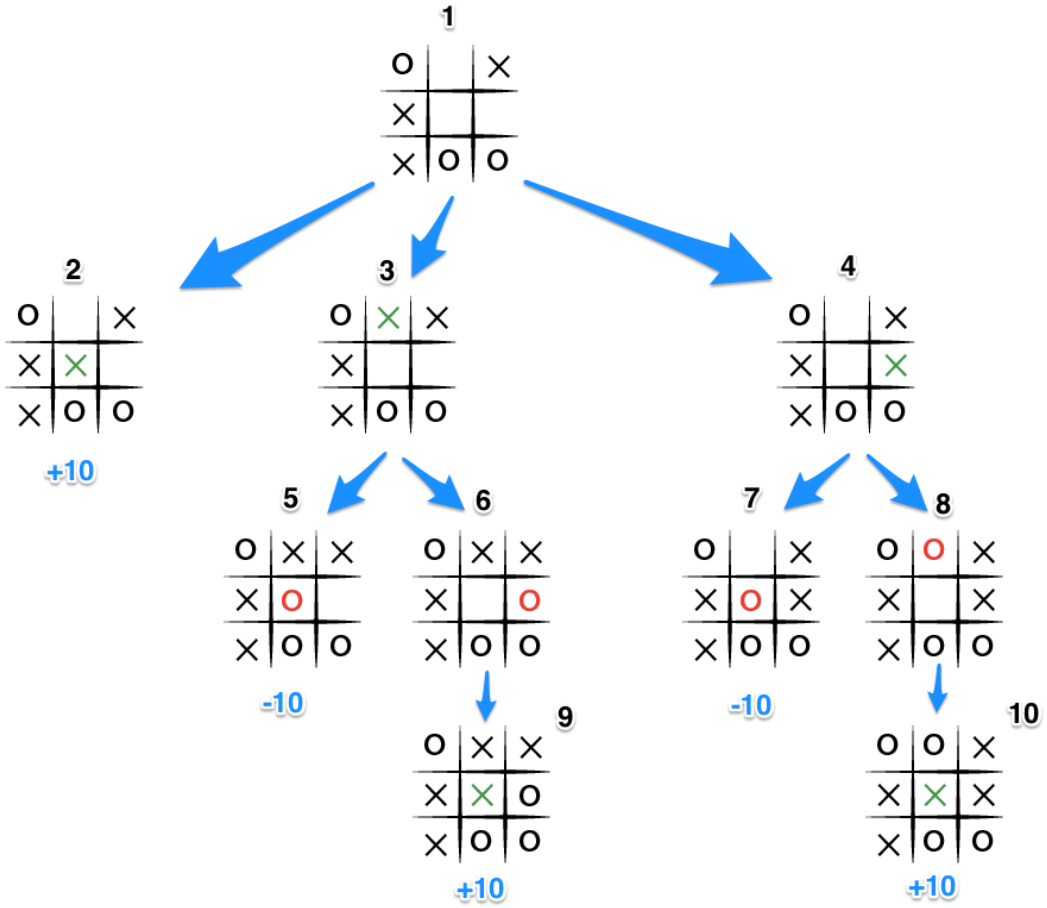
کار ما اینگونه عمل میکند که قرار است خانه هایی که روح نزدیک آن ها هست را امتیاز منفی نسبت بدهیم و خانه هایی که غذا نزدیک آن ها است را امتیاز مثبت. اولویت را فرار از گوست می گذاریم

از خط 86 تا 96 ما بدنبال پیدا کردن نزدیک ترین فذا هستیم و به هر خونه امتیاز قرینه فاصله‌اش از غذا رومیدهیم و اینگونه ارزش هر خونه را تایین میکنیم.

از خط ۹۹ به بعد اگر فاصله خانه‌ای از گوشت کمتر از دو بود یعنی ان خانه خطرناک است پس امتیاز منفی زیادی میدهیم به ان خانه

فصل سوم: الگوریتم مینی ماکس(minimax)

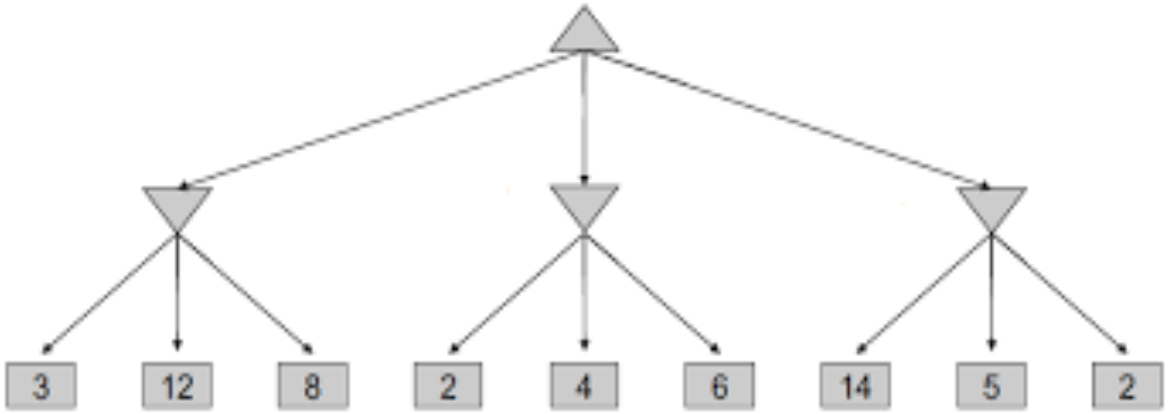
در این الگوریتم ما یک درخت از تمام حرکت های قابل رفتن از تمام بازیکن ها داریم و معمولا دو طرف دعوا داریم که یک تیم قصد پایین آوردن امتیاز بازی را دارد و تیم دیگر قصد بالا آوردن امتیاز را دارد و اسم این الگوریتم هم از روی همین روند انتخاب شده است.
حالا همانطور که گفته شد اول از همه باید برای تمام استیت هامون ما توانایی ارزش گذاری را داشته باشیم نسبت به حرکتی که انجام میدهیم. اما چگونه؟
برای مثال برای بازی دوز در نظر بگیریم.



فرض کنید ما بازیکن ایکس هستیم و بین جاهای موجود میخواهیم انتخاب کنیم و میخواهیم بازی حریف هم پیش بینی کنیم.
در لایه دوم میبینیم که ما انتخابی داریم که با انتخاب آن بازی تمام میشود و ما برنده ولی حالت های دیگر را ما محاسبه کردیم.
الگوریتم بعدی که میخوانیم مارا کمک میکند که این محاسبات اضافه را نکنیم.

این الگوریتم همیشه فرض میکند که حریف همیشه بهترین بازی را خواهد داشت و برای بازی مثل پک من وقتی در تله میوفتد به سمت حریف میرود که با بیشترین امتیاز ببازد و زمان هدر ندهد. اما الگوریتمی بعدا میخوانیم که با احتمال کار میکند و احتمال این که حریف همیشه بهترین بازی را نکند میدهد.

بیایید جور دیگری بررسی کنیم.



در این درخت مثلث های رو به بالا عاملی است که هدف بیشینه کردن امتیاز را دارد و مثلث های رو به پایین هدف کمینه کردن را دارد.

مربع های اخر وضعیتی هست که بازی تمام شده و به این وضعیت ترمینال میگوییم. بازیکن کمینه بین انتخاب هایی که دارد کمترین را انتخاب میکنید یعنی از 9 تا عدد اخر عدد های 3، 2 و 2 انتخاب میشوند و به لایه بالایی میروند. خب حالا نوبت بازیکن بیشینه است و بین عدد های انتخاب شده باید بیشترین را بر میدارد یعنی عدد 3 و با این وضعیت حرکتی که به عدد 3 ختم میشود را انجام میدهد.

پیاده سازی:

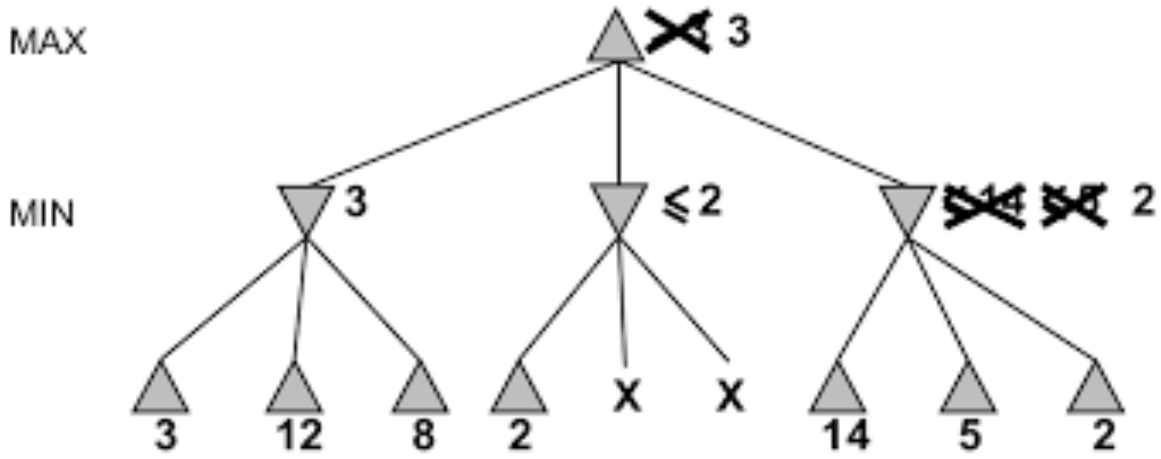
```
        "*** YOUR CODE HERE ***"
        def maxLayer(state, depth):
            #if the game is over or the depth is 0, return the score and the action
            if state.isWin() or state.isLose() or depth == 0:
                return self.evaluationFunction(state), None

            bestScore = float("-inf")

            #for each action, get the score and the action
            for action in state.getLegalActions(0):
                tmp = minLayer(state.generateSuccessor(0, action), depth, 1)
                if tmp > bestScore:
                    bestScore = tmp
                    bestAction = action
            return (bestScore, bestAction)
        def minLayer(state, depth, agentIndex):
            #if the game is over or the depth is 0, return the score
            if state.isWin() or state.isLose() or depth == 0:
                return self.evaluationFunction(state)

            bestScore = float("inf")

            #for each action, get the score
            for action in state.getLegalActions(agentIndex):
                if agentIndex == state.getNumAgents() - 1:
                    bestScore = min(bestScore, maxLayer(state.generateSuccessor(agentIndex, action), depth - 1)[0])
                else:
                    bestScore = min(bestScore, minLayer(state.generateSuccessor(agentIndex, action), depth, agentIndex + 1))
            return bestScore

        #return the action
        return maxLayer(gameState, self.depth)[1]
```

فصل چهارم : الگوریتم الفا بتا

این الگوریتم همان الگوریتم قبلی است فقط کار محاسباتی مارا کمتر میکند و درخت محاسبه مارا کوچک تر میکند.

این الگوریتم همان مینی ماکس هست فقط مسیر های اضافه را حذف میکند. اینجوری که طی تحلیل پی میبرد که مسیر در حال پردازش قطع به یقین مسیر انتخاب نهایی نخواهد یود و مسیر بهتری از این وجود دارد و دیگر آن مسیر را ادامه نمیدهد.

به مثال زیر توجه کنید:



در این مثال در لایه دوم نود سمت چپی تمام محاسبات را انجام میدهد و به عدد سه میرسد. ما میدانیم که نود پدر عدد 3 قرار است که بیشترین عدد را انتخاب کند. در نودهای برادر عدد سه ما قرار است یک سری از مسیر هارا نادیده بگیریم اما چگونه؟
ما میدانیم تنها باید عدد بزرگ تر از سه پیدا کنیم که برای ما اهمیت داشته باشد. پس در نود بعدی که ما عدد 2 را میبینیم میدانیم این عدد از تمامی عدد های بعدی یا بزرگ تر است یا کوچک تر . اگر کوچک تر باشد پس به عنوان مینیم انتخاب شده پس اهمیتی ندارد. اگر از همه بزرگ تر باشد پس عدد کمتری به عنوان مینیم انتخاب میشود پس همچنان برای ما نیست.
در این مسئله دو عدد با نام ها الفا و بتا داریم که به عنوان یک بازه ای که برای ما اهمیت دارد وجود دارد که اگر عددی از بتا کوچک تر ویا از الفا بزرگ تر باشد آن مسیر را حذف میکنیم.
پیاده سازی:

```
        "*** YOUR CODE HERE ***"
        def maxLayer(state, depth, alpha = float("-inf"), beta = float("inf")):
            if state.isWin() or state.isLose() or depth == 0:
                return self.evaluationFunction(state), "STOP"
            bestScore = float("-inf")
            #different from minimax, we need to pass alpha and beta to the next layer
            alphaToPass = alpha
            for action in state.getLegalActions(0):
                #condition to prune
                if bestScore > beta:
                    return (bestScore, action)
                alphaToPass = max(alphaToPass, bestScore)
                tmp = minLayer(state.generateSuccessor(0, action), depth, 1, alphaToPass, beta)
                if tmp > bestScore:
                    bestScore = tmp
                    bestAction = action
            return (bestScore, bestAction)
        def minLayer(state, depth, agentIndex, alpha = float("-inf"), beta = float("inf")):
            if state.isWin() or state.isLose() or depth == 0:
                return self.evaluationFunction(state)
            bestScore = float("inf")
            betaToPass = beta
            for action in state.getLegalActions(agentIndex):
                #condition to prune
                if bestScore < alpha:
                    return bestScore
                betaToPass = min(betaToPass, bestScore)
                if agentIndex == state.getNumAgents() - 1:
                    bestScore = min(bestScore, maxLayer(state.generateSuccessor(agentIndex, action), depth - 1, alpha, betaToPass)[0])
                else:
                    bestScore = min(bestScore, minLayer(state.generateSuccessor(agentIndex, action), depth, agentIndex + 1, alpha, betaToPass))
            return bestScore
        return maxLayer(gameState, self.depth)[1]
        #util.raiseNotDefined()
```

فصل پنجم: الگوریتم اکسپکتی مکس

این الگوریتم همان الگوریتم مینی ماکس است با تفاوت این که الگوریتم مینی ماکس فرض میکرد که رقیب بهترین حرکت خود را انجام میدهد. این الگوریتم به خود شانس اینکه رقیب جایی خطا کند را میدهد.
به این صورت که در درخت محاسبه به جای مینیمم گرفتن میانگین میگیریم از مجموعه کار هایی که میتواند انجام دهد.

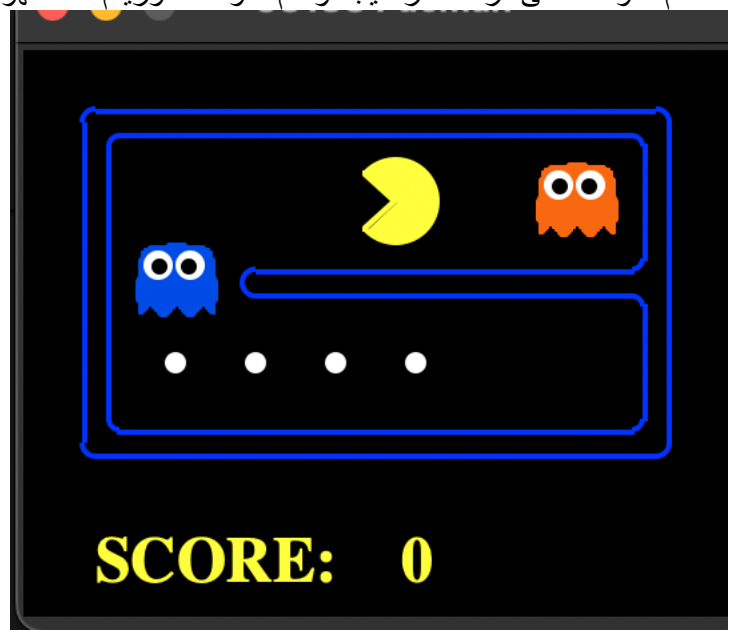
پیاده سازی:

```
255         "*** YOUR CODE HERE ***"
256         def maxLayer(state, depth):
257             if state.isWin() or state.isLose() or depth == 0:
258                 return self.evaluationFunction(state), "STOP"
259             bestScore = -100000000
260             for action in state.getLegalActions(0):
261                 tmp = minLayer(state.generateSuccessor(0, action), depth, 1)
262                 if tmp > bestScore:
263                     bestScore = tmp
264                     bestAction = action
265             return (bestScore, bestAction)
266         def minLayer(state, depth, agentIndex):
267             if state.isWin() or state.isLose() or depth == 0:
268                 return self.evaluationFunction(state)
269             bestScore = 0
270             for action in state.getLegalActions(agentIndex):
271                 #only difference from minimax is that we need to sum the scores
272                 if agentIndex == state.getNumAgents() - 1:
273                     bestScore += maxLayer(state.generateSuccessor(agentIndex, action), depth - 1)[0]
274                 else:
275                     bestScore += minLayer(state.generateSuccessor(agentIndex, action), depth, agentIndex + 1)
276             return bestScore / len(state.getLegalActions(agentIndex))
277         return maxLayer(gameState, self.depth)[1]
278         # util.raiseNotDefined()
```

میبینید که پیاده سازی قسمت پک من برای همان پک من است و تنها تفاوت در قسمت محاصبه روح ها است.
این الگوریتم از نظر زمانی کند تر از الفا بتا است برای اینکه ما میخواهیم در لایه روح میانگین بگیریم و نمیتوانیم بعضی از مسیر هارا حذف کنیم.
این الگوریتم برای زمانی مناسب است که ما بدانیم حریف به صورت بهینه عمل نمیکند و احتمال این که رندوم عمل کند را داریم.

فصل شیشم: بررسی کارکرد الگوریتم ها در شرایط خاص

الگوریتم مینیماکس و حرص الفا بتا در شرایطی که حریف بهترین کار را انجام میدهد بهینه هستند و بین خودشان نیز الگوریتم حرص الفا بتا به دلیل محاسبات کمتر بهتر است
در شرایط زیر مثال مهمی امده است
که هم تفاوت عمقی درخت در نتیجه و هم تفاوت الگوریتم ها مشهود است



در شرایط بالا اگر عمق درخت جست وجو برابر دو باشد پکمن مرگ خودش را نمیتواند پیش بینی کند چون فاصله گوست ابی برابر ۳ از پکمن است و به سمت گوست ابی میرود
ولی اگر عمق درخت ۳ باشد چون مرگ خود را قطعی میداند به سمت نزدیک ترین گوست میرود تا پوینت کمتر تلف کند.

اما چون گوست ها رنددوم حرکت میکنند پس امکان دارد گوست ابی به سمت پایین راست برود در این شرایط الگوریتم اکسپکتیماکس چون احتمال وجود این حالت رو نیز از پیش میدونست و میتوانست امتیاز بیشتری جمع اوری کند.