دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

پروژه دوم مبانی و کاربردهای هوش مصنوعی

نگارش
مهدیه سادات بنیس

استاد درس
دکتر مهدی جوانمردی

نیم سال دوم ۱۴۰۱

• بخش ۱: عامل عکس العمل

سوال: توضیح دهید که از هر کدام از پارامتر های دخیل درتابع ارزیابی چگونه استفاده کرده اید و هر کدام چگونه بر روی خروجی تاثیر میگذراند (تاثیر کدام یک از پارامتر ها بیشتر است؟).

در اینجا ما برای ارزیابی امتیاز حرکت های ممکن چهار عامل را بررسی میکنیم: ۱- فاصله با روح ها ۲- افزایش امتیاز ۳- فاصله با غذا ها ۴- مدت زمان ترس روح ها

فاصله با روح مهم ترین عامل برای ما است زیرا برخورد با روح میتواند بازی را به طور کل تمام کند پس خوب است در اولین گام بررسی کنیم اگر حرکت ما باعث برخورد با روح میشود کمترین امتیاز را بدهیم چون بدترین حالت است. در غیر این صورت کمترین فاصله روح ها را در امتیاز نهایی لحاظ میکنیم. اگر روح ها در حالت ترسیده نباشند هر چه فاصله بیشتر باشد بهتر است پس تاثیر مستقیم در امتیاز نهایی دارد.

در مرحله بعد بررسی میکنیم آیا امتیاز ما با انجام این افزایش می یابد یا نه. اگر امتیاز زیاد بشود اتفاق مثبتی است پس تابع ارزیابی باید مقدار مثبت تری بدهد تاثیر این عامل از دو عامل دیگر مهم تر است زیرا امتیاز همان لحظه را بیشتر میکند به همین علت در فرمول نهایی به آن ضریب بیشتری میدهیم.

عامل بعدی که میتواند در ارزیابی و امتیاز دهی به حرکت جدید موثر باشد تاثیر این حرکت در نزدیک شدن به غذاها است. برای این مورد فاصله نزدیکترین غذا را بدست میاوریم. هر چه این فاصله کمتر باشد بهتر است پس معکوس این مقدار را در امتیاز نهایی لحاظ میکنیم.

عامل آخر نیز مدت زمانی است که از حالت ترس روح ها مانده که این عامل هم باید رابطه مستقیم با امتیاز ارزیابی داشته باشد زیرا هر چه این زمان کمتر باشد بدتر است چون فرصت کمتری داریم پس با ضریب مثبت و به صورت مستقیم آن را در امتیاز نهایی اعمال میکنیم.

در نهایت این چهار عامل را به صورت وزن دار باهم ترکیب میکنیم و امتیاز نهایی را اعلام میکنیم. اهمیت امتیاز و فاصله تا روح ها را با توجه به ضرایب بیشتر در نظر گرفته ایم.

سوال: چگونه میتوان پارامتر هایی 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)

برای این کار کافیست پارامتر ها را در نهایت به صورت خطی با هم ترکیب کنیم (به نوعی میانگین وزن دار بگیریم) ضریب هر پارامتر نشان دهنده اهمیت و تاثیر آن پارامتر می باشد. پارامترهایی که مقادیرشان در یک راستا نیستند علامت ضریب آنها با هم مخالف است (یکی مثبت و دیگری منفی است) یا اینکه آن هایی که تاثیر مستقیم داند را خودشان و آن هایی که تاثیر عکس دارند معکوسشان را حساب کنیم (در واقع خطی نمیشود). برای مثال فاصله غذا تاثیر معکوس دارد(فاصله کمتر بهتر) اما فاصله تا روح تاثیر مستقیم (فاصله بیشتر بهتر)

سوال: راجع به نحوه پیاده‌سازی تابع evalutionfunction توضیح دهید همچنین توضیح دهید چرا امتیازی که برمیگردانید مناسب است و علت انتخاب نحوه محاسبه آن را بیان کنید.

در ابتدا مینیمم فاصله روح ها تا جایگاه فعلی (newPos) را بدست می آوردیم و در متغیر ghost_distance قرار میدهیم. سپس در صورتی که با روح برخورد میکردیم یا باید می ایستادیم کمترین امتیاز(5000-) را میدهیم.

سپس جمع مدت زمان ترس روح ها را محاسبه میکنیم که اگر بزرگتر از صفر بود پس مینیمم فاصله از روح ها معنی نخواهد داشت و مقدار آن را صفر قرار میدهیم.

در ادامه فاصله نزدیکترین غذا را حساب میکنیم.

فاصله های حساب شده در این قسمت از فاصله منهتن بدست آمده اند.

تابع ارزیابی:

$$2 * scare\_time/10 + (10/(nearest\_food + 1)) + (3 * ghost\_distance/10) + 4 * successorGameState.getScore()/10$$

با توجه به اهمیت هر کدام از مقدار های زمان ترس روح ها، مینیمم فاصله از روح ها و امتیاز مرحله بعدی به آن ها به ترتیب ضریب ۲و ۳ و ۴ اختصاص میدهیم (برتری و اهمیت هر کدام در جواب سوال اول ذکر شده است).

هر چه فاصله نزدیک ترین غذا کمتر باشد بهتر باشد پس رابطه آن با تابع ارزیابی باید معکوس باشد به همین علت آن را در مخرج قرار دادیم (با ۱ جمع کردیم که مقدار مخرج صفر نشود)

به دلیل زیاد بزرگ نشدن تابع ارزابی ۳ عامل را تقسیم بر ۱۰ کردیم و عامل فاصله نزدیک ترین غذا را در ۱۰ ضرب کردیم تا مقدار آن خیلی کوچک نشود که در تابع ارزیابی بی اثر شود.

```
ghost_distance = min([manhattanDistance(newPos, ghost_state.getPosition()) for ghost_state in newGhostStates])
        if action == Directions.STOP or ghost_distance <= 1:
            return -5000

        scare_time = sum(newScaredTimes)
        if scare_time > 0:
            ghost_distance = 0
        foods_distance = [manhattanDistance(newPos, food) for food in newFood.asList()]
        if foods_distance:
            nearest_food = min(foods_distance)
        else :
            nearest_food = 0
        return 2 * scare_time / 10 + (10 / (nearest_food + 1)) + (3 * ghost_distance / 10) + 4 *
successorGameState.getScore() / 10
```

• بخش ۲: مینیماکس

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:

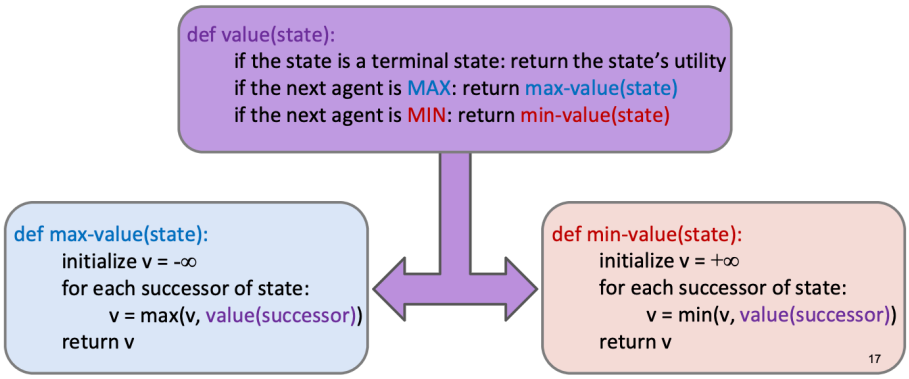
```
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
```

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

چون پکمن فرض میکند روح ها خصمانه ترین حالت ممکن عمل میکنند و اگر این اتفاق بیوفتد هیچ گاه به غذایی دست پیدا نمیکند و چون گذشت زمان نیز امتیاز را کم میکند ترجیح میدهد زودتر خودکشی کند تا اگر امتیازی بابت غذا نمیگیرد حداقل بابت عمرش امتیاز از دست ندهد

سوال: راجع به نحوه پیاده‌سازی min-max توضیح دهید.

با توجه به شبه کد توضیح داده در کلاس پیاده سازی را انجام میدهیم. با این تفاوت که ما به تعداد روح ها لایه های min داریم و نمیتوانیم لزوما یکی در میان بین max , min جا به جا شویم به همین علت لازم است وقتی در عمق min قرار داریم بررسی کنیم عمق بعدی چه نوعی است مربوط به روح است یا پکمن اگر مربوط به روح بود دوباره min اجرا میکنیم در غیر این صورت(در نوبت آخرین روح باشیم) max را اجرا میکنیم. برای شروع الگوریتم از ریشه هم باتوجه به اینکه پکمن در ریشه است کافیست همان تابع maxValue را اجرا کنیم با این تفاوت که حرکت بهینه را در متغیری ذخیره میکنیم تا به عنوان جواب برگردانیم.



در پیاده سازی نیست باتوجه به شبه کد توابع maxValue , minValue را پیاده سازی میکنیم. سپس برای گره ریشه میدانیم که مرحله بعدی min است پس از بین آن ها ماکسیمم را پیدا کرده و به عنوان حرکت بهینه ذخیره کرده و در خروجی بر میگردانیم.

```
Infinity = float('inf')
        def maxValue(gameState, depth):
            actions = gameState.getLegalActions(0)
            if not actions or depth == self.depth:   #Terminal Test
                return self.evaluationFunction(gameState)

            v = -Infinity
            for action in actions:
                successor = gameState.generateSuccessor(0, action)
                v = max (v, minValue(successor, depth + 1, 1))
            return v

        def minValue(gameState, depth, agentIndex):
            actions = gameState.getLegalActions(agentIndex)
            if not actions:   #Terminal Test
                return self.evaluationFunction(gameState)

            v = Infinity
            for action in actions:
                successor = gameState.generateSuccessor(agentIndex, action)
                # last ghost
```

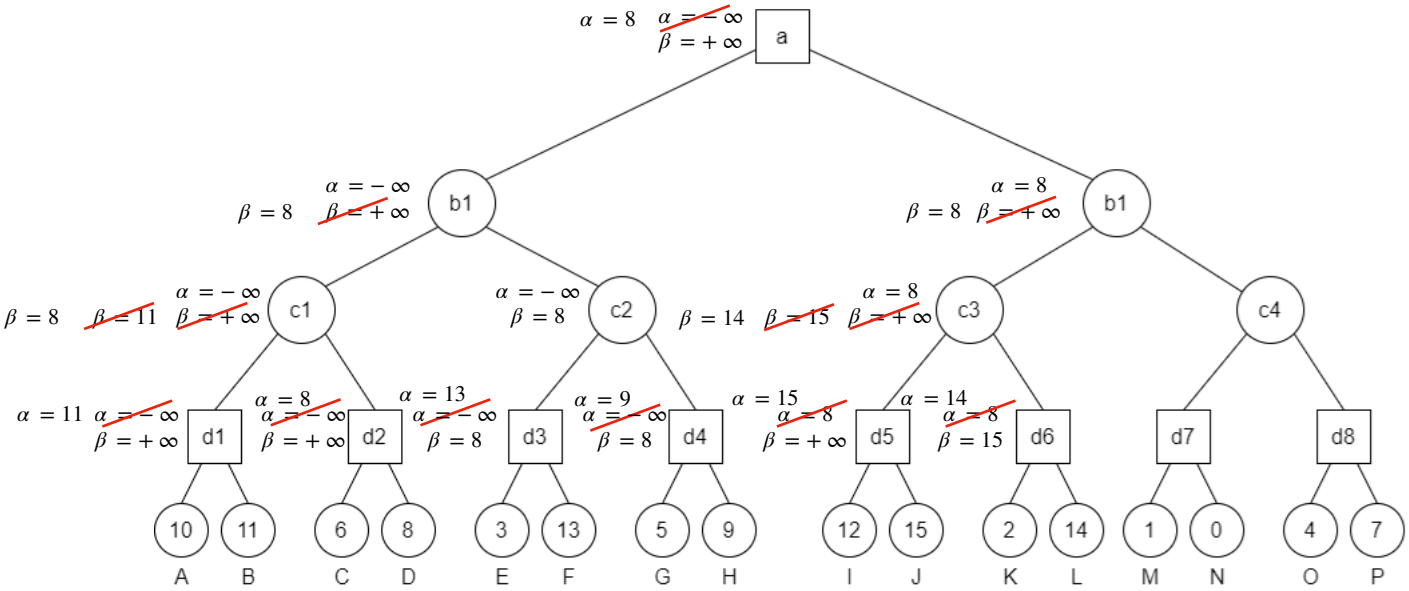
```
                if agentIndex == (gameState.getNumAgents() - 1):
                    v = min (v, maxValue(successor, depth))
                else:
                    v = min (v, minValue(successor, depth, agentIndex + 1))
            return v

        # Root level action
        actions = gameState.getLegalActions(0)
        v = -Infinity
        bestAction = actions[0]
        for action in actions:
            successor = gameState.generateSuccessor(0, action)
            # Next level is a min level
            newV = minValue(successor, 1, 1)
            # Choosing the action which is Maximum of the successors.
            if newV > v:
                bestAction = action
                v = newV
        return bestAction
```

• بخش ۳: هرس آلفا-بتا

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 ۲b شود و یا به سمت چپ حرکت کرده و وارد زیر درخت ۱b 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؟



حرکت بعدی پکم باید به سمت چپ باشد چون مقدار ماکسیمم در ریشه از سمت چپ میاید.

سوال: آیا در حالت کلی 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 خلاصه دلیل خودتان را توضیح دهید.

در هرس آلفا-بیتا مقدار بدست آمده در ریشه تغییر نمیکند زیرا در واقع فرآیند هرس ما به نوعی است که نتیجه نهایی اشتباه نشود اما در گره های میانی ممکن است این اتفاق بیوفته زیرا ما لزوما تاثیر تمام گره ها را بررسی نمیکنیم و اگر گره ای کران محدود کننده را تغییر ندهد آن را هرس میکنیم در نتیجه ممکن است مقدار متفاوتی در گره های میانی داشته باشیم.

سوال: نحوه پیاده سازی کدهای الگوریتم هرس الفا-بتا را توضیح دهید. در چه زمانی از این الگوریتم نمیتوانیم استفاده کنیم؟

مانند شبه کد آورده شده پیاده سازی را انجام میدهیم با این تفاوت که مانند قسمت قبل با توجه به اینکه به ازای هر روح یک لایه min داریم و ممکن است بیش از یک روح داشته باشیم در تابع max چک میکنیم عمق بعدی مربوط به کدام عامل است. همچنین مانند قسمت قبل برای شروع الگوریتم از ریشه درخت کافیست همان maxValue را انجام بدهیم با این تفاوت که حرکت برگزیده را ذخیره کنیم و به عنوان جواب نهایی باز گردانیم.

هرس آلفا بتا تکنیکی است که برای کاهش تعداد گره هایی که باید توسط الگوریتم مینیمکس ارزیابی شوند استفاده می شود. با این حال، شرایطی وجود دارد که در آن ها نمی توان از هرس آلفا بتا به طور موثر یا اصلا استفاده کرد. در اینجا چند سناریو وجود دارد که در آنها هرس آلفا بتا ممکن است کارساز نباشد:

- هنگامی که درخت بازی بسیار کوچک است: اگر درخت بازی بسیار کوچک باشد، هرس آلفا-بتا ممکن است هیچ بهبود قابل توجهی در زمان جستجو ایجاد نکند، و ممکن است به سادگی تمام گره های درخت را ارزیابی کنیم.

- وقتی فاکتور انشعاب زیاد است: هرس آلفا بتا زمانی بهترین کار را دارد که ضریب انشعاب درخت بازی نسبتاً کم باشد. اگر فاکتور انشعاب زیاد باشد، ممکن است تعداد زیادی گره برای ارزیابی وجود داشته باشد و هرس آلفا بتا ممکن است نتواند زمان جستجو را به میزان قابل توجهی کاهش دهد.

- وقتی تابع ارزیابی پیچیده است: اگر تابع ارزیابی که برای تخصیص مقادیر به گره‌ها در درخت بازی استفاده می‌شود پیچیده یا از نظر محاسباتی گران باشد، هرس آلفا-بتا ممکن است مزایای قابل‌توجهی نداشته باشد. در چنین مواردی، ارزیابی تمام گره های درخت به جای هرس کردن برخی از آنها ممکن است کارآمدتر باشد.

- وقتی بازی غیر قطعی است: هرس آلفا-بتا فرض می‌کند که بازی قطعی است، به این معنی که نتیجه هر حرکت قطعی است. اگر بازی غیر قطعی باشد، هرس آلفا-بتا ممکن است نتواند نتیجه هر حرکت را به طور دقیق پیش بینی کند و ممکن است مزایای قابل توجهی به همراه نداشته باشد.

```
def getAction(self, gameState):
        """
        Returns the minimax action using self.depth and self.evaluationFunction
        """
        "*** YOUR CODE HERE ***"
        Infinity = float('inf')
        def minValue(state, depth, agentIndex, a, b):
            actions = state.getLegalActions(agentIndex)
            if not actions:
                return self.evaluationFunction(state)
            v = Infinity
            for action in actions:
                successor = state.generateSuccessor(agentIndex, action)
                # last ghost
                if agentIndex == (state.getNumAgents() - 1):
                    v = min (v, maxValue(successor, depth, a, b))
                else:
                    v = min (v, minValue(successor, depth, agentIndex + 1, a, b))
                if v < a:
                    return v
                b = min(b, v)
            return v
        def maxValue(state, depth, a, b):
            actions = state.getLegalActions(0)
            if not actions or depth == self.depth:
                return self.evaluationFunction(state)
            v = -Infinity

            for action in actions:
                successor = state.generateSuccessor(0, action)
                v = max(v, minValue(successor, depth + 1, 1, a, b))
                if v > b:
                    return v
                a = max(a, v)
            return v
        # Root level action
        actions = gameState.getLegalActions(0)
        v = -Infinity
        a = -Infinity
        b = Infinity
        bestAction = actions[0]
        for action in actions:
            successor = gameState.generateSuccessor(0, action)
            # Next level is a min level. Hence calling min for successors of the root.
            newV = minValue(successor, 1, 1, a, b)
            # Choosing the action which is Maximum of the successors.
            if newV > v:
                bestAction = action
                v = newV

            if newV > b:
                return bestAction
            a = max(a,newV)
        return bestAction
```

• بخش ۴: مینیماکس احتمالی

سوال: همانطور که در سوال دوم اشاره شد روش مینیماکس درموقعیتی 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 درستور این گزاره را نشان دهید.همچنین دلیل این تفاوت در عملکرد مینیماکس و مینیماکس احتمالی را توضیح دهید.

همان طور که مشاهده میکنیم در نقشه ای که پکمن در دام گیر میکند با minimax پکمن فرض میکند رقیب ها خصمانه ترین حرکت ممکن را انجام میدهد به همین دلیل تصمیم به خودکشی میگیرد تا امتیاز کمتری از دست دهد و هر ۱۰ بازی را می بازد.اما رقیب ها لزوما به خصمانه ترین حالت عمل نمیکنند و زمانی که از الگوریتم expectimax استفاده میشود عامل پکمن احتمالی هم میدهد که اگر خصمانه ترین حالت ممکن رخ نداد چه اتفاقی میوفتد و سعی میکند راهی پیدا کند که اگر شرایط کمی بهتر از خصمانه ترین حالت ممکن شد ببرد به همین دلیل میتواند تعدادی از باری ها را حتی در نقشه دام ببرد چون حرکت عامل های رقیب احتمالی است و در بازی هایی که همیشه خصمانه ترین حرکت انجام نمیشود احتمال برد پکمن وجود دارد زیرا در تصمیم گیری اش این احتمال را مدنظر گرفته و بعضی جا ها ریسک میکند و به سمت غذا میرود و امیدوار است تا رقیب خصمانه عمل نکند و به جای آن امتیاز ناشی از رسیدن به غذا را دریافت کند.

```
(base) mahdieh@mahdiehs-MacBook-Pro AI_P2_SP23 % 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 died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Average Score: 118.4
Scores:        532.0, 532.0, 532.0, -502.0, -502.0, -502.0, 532.0, 532.0, 532.0, -502.0
Win Rate:      6/10 (0.60)
Record:        Win, Win, Win, Loss, Loss, Loss, Win, Win, Win, Loss
(base) mahdieh@mahdiehs-MacBook-Pro AI_P2_SP23 % 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
(base) mahdieh@mahdiehs-MacBook-Pro AI_P2_SP23 %
```

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 ارزیان 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؟

انتخاب هر کروموزوم در این الگوریتم به صورت احتمالی بوده به نحوی که احتمال انتخاب هر کروموزوم متانسب با fitness آن کروموزوم می باشد یعنی برای کروموزوم x احتمال انتخاب به صورت زیر خواهد بود:

$$p_x = \frac{fitness_x}{\sum fitness_x}$$

با توجه به اینکه انتخاب احتمالی در لایه مربوط به روح ها استفاده میشود معیار ما (fitness در فرمول بالا) میتواند میزان امتیاز پکمن یا فاصله روح و پکمن باشد. برای بررسی بیشتر از یک حالت خوب است از چیزی همانند

درخت expectimax استفاده کرد با این تفاوت که احتمال ها در لایه های مربوط به رقیب به کمک الگوریتم رولت ویل محاسبه شود.

سوال: در سناریوهای احتمالاتی که حرکات ارواح کامل اقطعی نیستند، استراتژی بهینه برای یک عامل Pac-Man با استفاده از الگوریتم کمینه احتمالی یا Expectimax چیست؟ این استراتژی چه تفاوتی با استراتژی در سناریوهای قطعی دارد؟ تحلیل پیچیدگی استراتژی بهینه و مقایسه آن با استراتژی در سناریوهای قطعی. مدل های احتمالی مختلف و تاثیر آنها بر استراتژی بهینه را در نظر بگیرید.

در سناریوهای احتمالی که حرکات ارواح کاملاً قطعی نیست، می توان از الگوریتم Expectimax برای یافتن استراتژی بهینه برای یک عامل Pac-Man استفاده کرد. الگوریتم Expectimax تعمیم یافته الگوریتم minimax است که توزیع احتمال را بر روی نتایج احتمالی هر اقدام در نظر می گیرد.

در مورد Pac-Man، الگوریتم Expectimax را می توان برای محاسبه مطلوبیت مورد انتظار هر عمل با در نظر گرفتن توزیع احتمال بر روی حرکات احتمالی ارواح استفاده کرد. استراتژی بهینه برای عامل Pac-Man این است که اقدامی را با بالاترین سود مورد انتظار انتخاب کند.

استراتژی در سناریوهای احتمالی با استفاده از الگوریتم Expectimax با استراتژی در سناریوهای قطعی، که در آن حرکات ارواح به طور کامل شناخته شده است، متفاوت است. در سناریوهای قطعی، استراتژی بهینه برای عامل Pac-Man استفاده از الگوریتم minimax برای یافتن بهترین اقدام برای انجام است، با این فرض که ارواح همیشه حرکت بهینه را برای به حداقل رساندن سودمندی Pac-Man انجام می دهند.

با این حال، در سناریوهای احتمالی، حرکات ارواح نامشخص است و عامل Pac-Man باید این عدم قطعیت را در هنگام انتخاب اقدامات خود در نظر بگیرد. الگوریتم Expectimax به عامل Pac-Man اجازه می دهد تا بر اساس احتمالات نتایج مختلف تصمیم گیری کند، نه اینکه فرض کند ارواح همیشه حرکت بهینه را انجام می دهند.

پیچیدگی استراتژی بهینه برای یک عامل Pac-Man با استفاده از الگوریتم Expectimax Expectimax در سناریوهای احتمالی به عوامل مختلفی بستگی دارد، از جمله اندازه صفحه بازی، تعداد ارواح، تعداد حرکات ممکن برای هر روح، و تعداد حالت های ممکن برای هر حرکت ارواح.

در مقایسه با سناریوهای قطعی، استراتژی بهینه با استفاده از الگوریتم Expectimax در سناریوهای احتمالی می‌تواند پیچیده‌تر و از نظر محاسباتی فشرده‌تر باشد. این به این دلیل است که عامل Pac-Man باید توزیع احتمال را بر روی حرکات احتمالی ارواح در نظر بگیرد، که می‌تواند منجر به فضای جستجوی بسیار بزرگ‌تر و زمان‌های محاسباتی طولانی‌تر شود.

تأثیر مدل‌های احتمالی مختلف بر استراتژی بهینه نیز می‌تواند قابل توجه باشد. به عنوان مثال، اگر عامل Pac-Man اطلاعات کاملی در مورد حرکات ارواح داشته باشد، اما ارواح به طور تصادفی با توزیع یکنواخت حرکت می کنند، استراتژی بهینه ممکن است شامل اولویت دادن به حرکات ایمن و اجتناب از مناطقی از صفحه باشد که عامل Pac-Man در آن قرار دارد. به احتمال زیاد توسط ارواح به دام افتاده است.

از سوی دیگر، اگر حرکات ارواح تحت تأثیر یک مدل احتمالی پیچیده‌تر، مانند فرآیند تصمیم مارکوف یا فرآیند تصمیم‌گیری مارکوف تا حدی قابل مشاهده باشد، استراتژی بهینه ممکن است شامل در نظر گرفتن عوامل دیگری مانند Pac-Man باشد. حرکات و مشاهدات خود عامل و الگوهای رفتاری و ترجیحات ارواح.

به طور کلی، پیچیدگی استراتژی بهینه با استفاده از الگوریتم Expectimax در سناریوهای احتمالی را می توان با استفاده از روش های اکتشافی یا سایر فرضیات ساده کننده برای کاهش فضای جستجو و زمان محاسبات کاهش داد. با این حال، معامله این است که استراتژی حاصل ممکن است همیشه بهینه نباشد، و ممکن است در محیط‌های پیچیده‌تر یا غیرقابل پیش‌بینی‌تر به خوبی عمل نکند.

سوال: درسناریوهای احتمالاتی که عامل Pac-Man فقط مشاهدات جزئی از وضعیت بازی دارد، چگونه عامل می تواند از probabilistic minimax یا Expectimax احتمالی برای تصمیم‌گیری بهینه استفاده کند؟ چگونه عامل می تواند باور خود را در مورد وضعیت بازی بر اساس مشاهدات جدید به روزکند؟ مبادلات بین اکتشاف و بهره برداری را در سناریوهای مشاهده جزئی تجزیه و تحلیل کنید و آنها را با مبادلات در سناریوهای مشاهده کامل مقایسه کنید. مدل های مختلف مشاهده جزن و تاثیر آنها بر فرآیند تصمیم گیی را در نظر بگیرید.

در سناریوهای احتمالاتی که عامل Pac-Man فقط مشاهدات جزئی از وضعیت بازی دارد، عامل می تواند از الگوریتم Expectimax با حالت های اعتقادی برای تصمیم گیری بهینه استفاده کند. حالت باور نشان دهنده دانش فعلی یا عدم اطمینان عامل در مورد وضعیت واقعی بازی، بر اساس مشاهدات و اقدامات قبلی آن است. عامل می تواند حالت اعتقاد خود را بر اساس مشاهدات جدید به روز کند و از آن به عنوان ورودی الگوریتم Expectimax برای محاسبه مطلوبیت مورد انتظار هر عمل استفاده کند.

فرآیند به‌روزرسانی حالت باور شامل استفاده از قانون بیز برای محاسبه توزیع پسین روی حالت‌های احتمالی بازی، با توجه به مشاهده فعلی و حالت اعتقاد قبلی است. این نیاز به مدلی از احتمالات مشاهده دارد، که احتمال مشاهده هر مشاهده ممکن را با توجه به هر حالت بازی ممکن مشخص می کند.

مبادلات بین اکتشاف و بهره برداری در سناریوهای مشاهده جزئی شبیه به سناریوهای مشاهده کامل است، اما به دلیل عدم اطمینان عامل در مورد وضعیت واقعی بازی، آنها پیچیده تر هستند. عامل باید تمایل خود را برای کشف اقدامات جدید و به دست آوردن اطلاعات بیشتر در مورد وضعیت بازی با نیاز خود به بهره برداری از دانشی که از قبل برای به حداکثر رساندن مطلوبیت مورد انتظار خود دارد، متعادل کند.

مدل‌های مشاهده‌ای مختلف می‌توانند تأثیر قابل‌توجهی بر فرآیند تصمیم‌گیری داشته باشند. برای مثال، اگر مدل مشاهده پر سر و صدا یا غیرقابل اعتماد باشد، ممکن است عامل نیاز به کاوش بیشتری داشته باشد تا عدم اطمینان خود را در مورد وضعیت واقعی بازی کاهش دهد. اگر مدل مشاهده آموزنده باشد یا با وضعیت واقعی بازی ارتباط زیادی داشته باشد، ممکن است عامل بتواند از دانش خود به طور مؤثرتری بهره برداری کند.

به طور کلی، استفاده از حالت های باور و الگوریتم Expectimax در سناریوهای مشاهده جزئی می تواند عملکرد تصمیم گیری عامل را بهبود بخشد، اما با قیمت افزایش پیچیدگی محاسباتی و نیاز به مدل های با مشاهده دقیق دارد.

• بخش ۵: تابع ارزیابی

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

در این بخش تابع ارزیابی قرار داده شده همانی است که در مورد یک نوشته شد اما به جای استفاده از successorsGameState از خود currentGameState استفاده شده است. البته برخی از ضریب ها تغییر کرده و همچنین عامل دیگری (فاصله کپسول ها که تاثیر عکس دارد) نیز در تصمیم گیری دخیل شده است. عملکرد این تابع سریع تر است زیرا برای حرکت های مختلف بررسی نمیشود و وابسته به حالت فعلی و قطعی است. همچنین معیار های دیگری نیز به این تابع اضافه کردیم.

سوال: طراحی بهینه یک تابع ارزیابی برای یک عامل Pac-Man چیست؟ آیا میتوان یک تابع ارزیابی طراحی کرد که بازی بهینه را در همه حالت‌های بازی تضمین کند؟ اگرنه، محدودیت‌های رویکرد عملکرد ارزیابی چیست؟ معاوضه بین دقت وکارایی محاسباتی در طراحی تابع ارزیابی را تجزیه و تحلیل کنید. طراحی های مختلف عملکرد ارزیابی و تاثیر آنها بر رفتار عامل را در نظر بگید.

طراحی بهینه یک عملکرد ارزیابی برای یک عامل Pac-Man باید اهداف و اولویت های اساسی بازی را منعکس کند. به طور معمول، تابع ارزیابی بر اساس عواملی مانند تعداد غذاهای خورده شده، فاصله تا نزدیک‌ترین روح، زمان باقی‌مانده و سایر اهداف مربوط به بازی، امتیازی را به هر حالت ممکن از بازی اختصاص می‌دهد.

طراحی عملکرد ارزیابی که گیم پلی بهینه را در همه حالت های بازی تضمین کند، ممکن نیست، زیرا رفتار بهینه ممکن است به ویژگی ها و اهداف خاص هر حالت بستگی داشته باشد. با این حال، یک عملکرد ارزیابی خوب طراحی شده می تواند منجر به عملکرد خوب در طیف گسترده ای از حالت ها و محیط های بازی شود.

محدودیت‌های رویکرد ارزیابی عملکرد شامل پتانسیل سوگیری یا ناقص بودن عملکرد ارزیابی است که منجر به رفتار غیربهینه یا غیرقابل پیش‌بینی در برخی موقعیت‌ها می‌شود. علاوه بر این، پیچیدگی محاسباتی تابع ارزیابی می تواند به یک گلوگاه مهم در فرآیند تصمیم گیری تبدیل شود، به خصوص اگر تابع بر محاسبات یا شبیه سازی های پیچیده متکی باشد.

مبادله بین دقت و کارایی محاسباتی در طراحی تابع ارزیابی به الزامات و محدودیت های خاص محیط بازی بستگی دارد. به طور کلی، عملکردهای ارزیابی دقیق تر و جامع تر ممکن است منجر به عملکرد بهتر شود، اما به قیمت افزایش پیچیدگی محاسباتی و زمان محاسبات طولانی تر. از سوی دیگر، توابع ارزیابی ساده‌تر و کارآمدتر از نظر محاسباتی ممکن است برخی از دقت و انعطاف‌پذیری را قربانی کنند، اما می‌توانند در محیط‌های بازی بزرگ یا پیچیده‌تر عملی‌تر و مقیاس‌پذیرتر باشند.

طرح‌های مختلف ارزیابی عملکرد می‌تواند تأثیر قابل‌توجهی بر رفتار عامل Pac-Man داشته باشد. به عنوان مثال، طرحی که بر بقا و اجتناب از ارواح تاکید دارد ممکن است منجر به رفتار محافظه‌کارانه‌تر و ریسک‌گریزتر شود، در حالی که طرحی که مصرف گلوله و شکار تهاجمی ارواح را در اولویت قرار می‌دهد ممکن است منجر به رفتار تهاجمی‌تر و پرخطرتر شود. انتخاب های طراحی خاص باید بر اساس تجزیه و تحلیل دقیق اهداف بازی و معاوضه بین معیارهای مختلف عملکرد باشد.

سوال: سه تابع ارزیابی مختلف را آزمایش کرده و نتیجه‌های آنان را گزارش کنید.همچنین نقاط قوت و ضعف هرکدام و علت کارآیی یا ناکارآمدی هرکدام را توضیح دهید دقت کنید که تمام توابع شما باید با امتیاز بالا(امتیازی که خودتان از این بخش گرفتید) این بخش را تمام کنند. در مقایسه تان از دقت، کارآیی محاسباتی و دیگر فاکتورهای لازم استفاده کنید.

```
return capsuleScore + scare_time + (10 / (nearest_food + 1)) + (3 * ghost_distance / 10) + 4 * currentGameState.getScore() / 10
```

```
(base) mahdieh@mahdiehs-MacBook-Pro AI_P2_SP23 % python autograder.py -q q5 --no-graphics
/Users/mahdieh/Desktop/aut/Term 8/CE/Principles & Applications of Artificial Intelligence/Project/2/AI_P2_SP23/autograder.py:17: Dep
recationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 5-3 at 3:47:36

Question q5
===========

Pacman emerges victorious! Score: 1111
Pacman emerges victorious! Score: 1051
Pacman emerges victorious! Score: 1330
Pacman emerges victorious! Score: 1363
Pacman emerges victorious! Score: 884
Pacman emerges victorious! Score: 1133
Pacman emerges victorious! Score: 1320
Pacman emerges victorious! Score: 742
Pacman emerges victorious! Score: 1177
Pacman emerges victorious! Score: 1242
Average Score: 1135.3
Scores:        1111.0, 1051.0, 1330.0, 1363.0, 884.0, 1133.0, 1320.0, 742.0, 1177.0, 1242.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q5/grade-agent.test (6 of 6 points)
***     1135.3 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q5: 6/6 ###


Finished at 3:47:40

Provisional grades
==================
Question q5: 6/6
------------------
Total: 6/6
```

```
return capsuleScore + scare_time + (5 / (nearest_food + 1)) + (3 * ghost_distance / 10) + 4 * currentGameState.getScore() / 10
```

```
(base) mahdieh@mahdiehs-MacBook-Pro AI_P2_SP23 % python autograder.py -q q5 --no-graphics
/Users/mahdieh/Desktop/aut/Term 8/CE/Principles & Applications of Artificial Intelligence/Project/2/AI_P2_SP23/autograder.py:17: Dep
recationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 5-3 at 3:54:59

Question q5
===========

Pacman emerges victorious! Score: 1166
Pacman emerges victorious! Score: 881
Pacman emerges victorious! Score: 864
Pacman emerges victorious! Score: 1202
Pacman emerges victorious! Score: 958
Pacman emerges victorious! Score: 1339
Pacman emerges victorious! Score: 1024
Pacman emerges victorious! Score: 1195
Pacman emerges victorious! Score: 1114
Pacman emerges victorious! Score: 1127
Average Score: 1087.0
Scores:        1166.0, 881.0, 864.0, 1202.0, 958.0, 1339.0, 1024.0, 1195.0, 1114.0, 1127.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q5/grade-agent.test (6 of 6 points)
***     1087.0 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q5: 6/6 ###
```

```
Finished at 3:55:09

Provisional grades
==================
Question q5: 6/6
------------------
Total: 6/6
```

```
return capsuleScore + 2 * scare_time + (10 / (nearest_food + 1)) + (3 * ghost_distance / 10) + 4 * currentGameState.getScore() / 10
```

```
(base) mahdieh@mahdiehs-MacBook-Pro AI_P2_SP23 % python autograder.py -q q5 --no-graphics
/Users/mahdieh/Desktop/aut/Term 8/CE/Principles & Applications of Artificial Intelligence/Project/2/AI_P2_SP23/autograder.py:17: Dep
recationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 5-3 at 3:56:58

Question q5
===========

Pacman emerges victorious! Score: 1251
Pacman emerges victorious! Score: 992
Pacman emerges victorious! Score: 1065
Pacman emerges victorious! Score: 1044
Pacman emerges victorious! Score: 1026
Pacman emerges victorious! Score: 1159
Pacman emerges victorious! Score: 1156
Pacman emerges victorious! Score: 1166
Pacman emerges victorious! Score: 1036
Pacman emerges victorious! Score: 1164
Average Score: 1105.9
Scores:        1251.0, 992.0, 1065.0, 1044.0, 1026.0, 1159.0, 1156.0, 1166.0, 1036.0, 1164.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q5/grade-agent.test (6 of 6 points)
***     1105.9 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q5: 6/6 ###


Finished at 3:57:09

Provisional grades
==================
Question q5: 6/6
------------------
Total: 6/6
```

همان طور که مشاهده میکنیم سه تابع ارزیابی در میانگین امتیاز دارای تفاوت هستند و تابع اول بیشتر از دوم میباشد زیرا که پارامتر سوم آن که نزدیکترین غذا است ضریب بیشتری دارد و در تابع ارزیابی نیست دارای اهمیت بالاتری می باشد و چونکه به آن ارزش بالاتری داده شده میانگین امتیاز نیست بالاتر است. همچنین مدت زمان اجرا اول ۴ ثانیه ، دوم ۱۰ ثانیه و سوم ۱۱ ثانیه است و علت آنکه اول از بقیه کمتر است ضریب پارامتر اول است که باعث میدهد زودتر نقطه بهینه انتخاب شود و همچنین زمان ترس روح ها نیز دارای ضریب کمتری است زیرا به نسبت اهمیت کمتری دارد.