محمدحسین صباغی ۹۹۲۳۰۴۴

گزارشکار پروژه ۲ هوش مصنوعی

۱)

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

فاصله از نزدیکترین روح را پیدا کرده و آنرا یه عامل مثبت در نظر گرفتیم که در تابع ضریب مثبت دارد.

فاصله از نزدیکترین غذا را پیدا کرده و آنرا بعنوان یک عامل منفی در نظر گرفته و ضریب منفی به آن دادیم.

همچنین تعداد غذا های باقی مانده و خورده شده را هم به عنوان عامل های منفی و مثبت در نظر گرفتیم و به عنوان ضریب برای عامل های قبل در نظر گرفتیم.

تاثیر دوری از روح خیلی نزدیک و تعداد غذا های باقی مانده از بقیه عوامل 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)

پارامتر هایی که خوب هستند (مانند فاصله از روح ها) را میتوان باضریب مثبت و پارامترهایی که بد هستند(مانند غذاهای باقی مانده) را با ضریب منفی در نظر میگیریم و با دادن ضریب به آنها سعی میکنیم ارزش آنهارا بالانس کنیم.

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 همانطور که واضح است فاصله تا نزدیک ترین غذا را محاسبه کرده که عاملی منفی است .

سپس روح هایی که غیب نشده اند مورد ارزیابی قرار میگیرند و هرکدام فاصلشان کمتر بود، فاصله آنرا در نظر میگیریم که عاملی مثبت است. البته اگر روح دور از ما بود اصلا آنرا در نظر نمیگیریم زیرا مارا تهدید نمیکند و بهتر است به خوردن غذاها بپردازیم.

و در آخر غذاهای باقی مانده و خورده شده را به عنوان ضریب برای عوامل در نظر میگیرم و با بالا و پایین کردن ضرایب و تست کردن آنها به حالتی میرسیم که بنظر بهتر است.

علت انتخاب آن مشخص بودن اهمیت بالای فاصله از غذاها و روح ها و غذایهای باقی مانده است و در نهایت بالا و پایین کردن ضرایب برای رسیدن به بهترین عملکرد.

۲)

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؟

زیرا پکمن در هر صورت میمیرد پس سعی میکند زودتر بمیرد تا به خاطر تایم زنده موندن امتیاز منفی بیشتری نگیرد

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

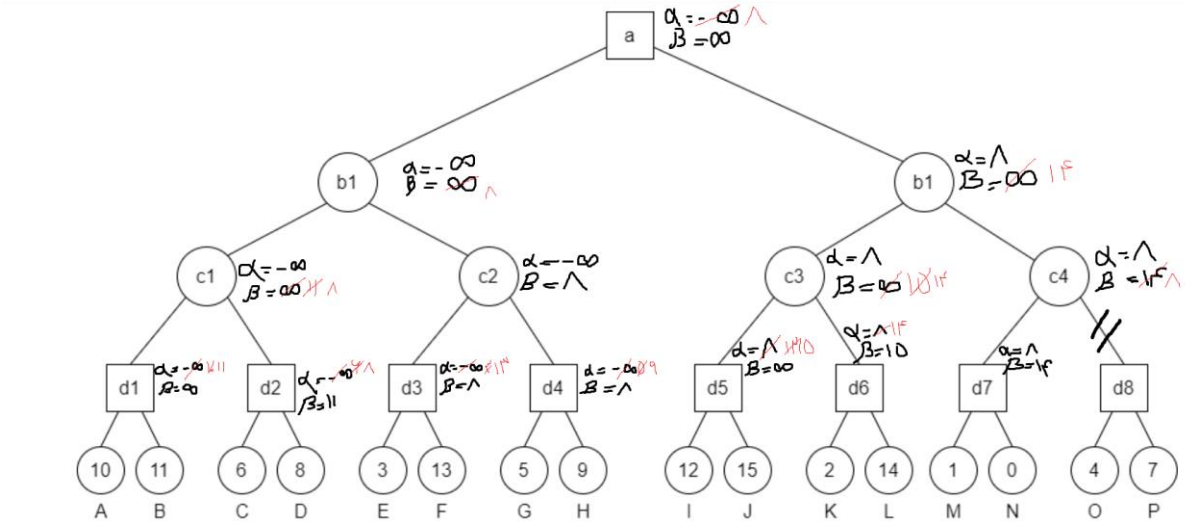
برای اینکار از دو تابع بازگشتی min و max استفاده شده است که ماکسیمم برای انتخاب های خود پکمن است و درو آن min صدا زده میشود تا حرکات روح ها اجرا شود.

هرجا روح ها تمام شوند یکی به عمق اضافه میشود و چک میشود اگر به عمق دلخواه رسیده باشیم دیگر ادامه ندهیم. اگر هم روح ها تمام شده باشند و به عمق دلخواه نرسیده باشیم دوباره با صدا زدن max به سراغ محاسبه حرکت پکمن میرویم.

در min کمترین امتیاز حرکت بعد برمیگردد و در max بیشترین امتیاز(در کنار حرکت مورد نیاز برای اینکار)

۳)

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 b۲ شود و یا به سمت چپ حرکت کرده و وارد زیر درخت b۱ 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؟



D8 هرس می‌شود زیرا d7 برابر ۱ شده است و اهمیتی ندارد که d8 از آن بزرگتر میشود یا کوچکتر. زیرا در هر صورت مقدار زیردرخت راست aکمتر از زیردرخت چپ آن میشود.

جواب چپ میشود.

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 گره‌های میانی چطور؟ به طور خلاصه دلیل خودتان را توضیح دهید.

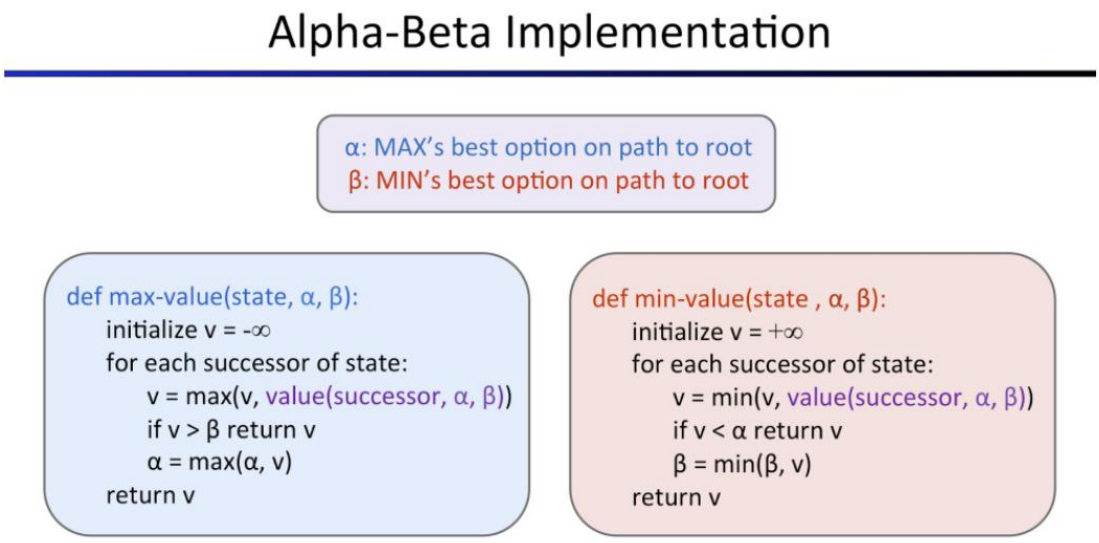
بطور کلی انتخابی که در ریشه میکند تغییر نمیکند ولی ممکن است امتیاز بدست آمده برای آن تغییر کند. زیرا ممکن است مطمئن شود مقدار امتیاز زیر درخت سمت راست بیشتر از زیر درخت چپ است و دیگر به کاوش کردن ادامه ندهد. در این حالت ممکن است امتیاز بیشتری در سمت راست باشد ولی اهمیتی برای ما ندارد صرفا دانستن اینکه سمت راست بهتر است برای ما کافی است.

برای گره های میانی هم امکان دارد انتخاب تغییر کند هم ممکن است مقدار امتیاز آن تغییر کند.ممکن است در حال کاوش زیر درخت سمت راست باشیم و با هرس آلفا بتا متوجه شویم که زیر درخت راست در هر صورت بهتر است. با اینکه نمیدانیم در ادامه کاوش در گره میانی کدام حرکت بهتر خواهد بود.

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

مانند الگوریتم ارایه شده در دستور کار دو فانکشن بازگشتی تعریف شده است که شرط خروج از آنها رسیدن به عمق دلخواه یا رسیدن به وضعیت برنده یا بازنده است. همچنین هرگاه مقدار آلفا یا بتا تغییر کند اکشن هم تغییر میکند.

ابن الگوریتم برای مینیماکس است و برای حالاتی که چیزی غیر از مینیمم یا ماکسیمم داشته باشیم کاربرد ندارد و نمیتواند استفاده شود. حتی در مینیماکس احتمالی هم نمیتواند استفاده شود زیرا به تمام برگ ها نیاز داریم.



```python
def maxFunc(self, gamestate, agentIndex, d, a, b):
    if gamestate.isWin():
        return ("STOP", self.evaluationFunction(gamestate))
    if gamestate.isLose():
        return ("STOP", self.evaluationFunction(gamestate))
    actions = gamestate.getLegalActions(agentIndex)
    v = -100000
    resAction = None
    for action in actions:
        successor = gamestate.generateSuccessor(agentIndex, action)
        v = max(v, self.minFunc(successor, 1, d + 1, a, b)[1])
        if v > b:
            return (resAction, v)
        temp = max(a, v)
        if temp != a:
            resAction = action
        a = temp
    return (resAction, v)
```

```python
def minFunc(self, gamestate, agentIndex, d, a, b):
    if gamestate.isLose():
        return ("STOP", self.evaluationFunction(gamestate))
    if gamestate.isWin():
        return ("STOP", self.evaluationFunction(gamestate))
    if agentIndex == gamestate.getNumAgents():
        if d == self.depth:
            return ("STOP", self.evaluationFunction(gamestate))
        else:
            return self.maxFunc(gamestate, 0, d, a, b)
    actions = gamestate.getLegalActions(agentIndex)
    v = 100000
    resAction = None
    for action in actions:
        successor = gamestate.generateSuccessor(agentIndex, action)
        v = min(v, self.minFunc(successor, agentIndex + 1, d, a, b)[1])
        if v < a:
            return (resAction, v)
        temp = min(b, v)
        if temp != b:
            resAction = action
        b = temp
    return (resAction, v)
```

(۴

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
```

```
G:\AI P2\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

G:\AI P2\AI_P2_SP23>python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
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: 531
Pacman emerges victorious! Score: 531
Pacman emerges victorious! Score: 531
Pacman died! Score: -502
Average Score: -88.7
Scores:        -502.0, -502.0, -502.0, -502.0, -502.0, 532.0, 531.0, 531.0, 531.0, -502.0
```

دلیلش اینست که مینیماکس همیشه بدترین حالت را در نظر میگیرد درحالی که روح ها همیشه بدترین حالت را اجرا نمیکنند. بنابراین با در نظر گرفتن بدترین حالت که همان مرگ است ترجیح میدهد سریعتر ببازد تا امتیاز منفی کمتری بگیرد.

ولی در مینیماکس احتمالی پکمن میداند که روح ها ممکن است بدترین حالت را اجرا نکنند و در نتیجه سعی میکند از باختن جلوگیری کند تا شاید روح ها به سمت او نیامدند و توانست بازی را ببرد.

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؟

انتخاب شانسی میباشد ولی احتمال انتخاب هر کروموروم متفاوت است و بر اساس سازگاری آن با شرایط مطلوب است. یعنی هرچه یک کروموزوم شانس موفقیت بیشتر داشته باشد احتمال بیشتری برای انتخاب شدن دارد ولی در نهایت بصورت شانسی انتخاب میشود.

میتوان احتمال انتخاب هر اکشن را برابر نسبت امتیاز آن اکشن به مجموع امتیاز های همه ی اکشن ها در نظر گرفت و سپس با احتمال های بدست آمده بصورت شانسی ولی با احتمال متفاوت انتخاب کنیم. پس از انتخاب یک حالت، آن اکشن را کنار میزاریم و بین باقی اکشن ها بصورت شانسی و بر اساس احتمالات آنها(که متناسب با امتیاز آن اکشن تقسیم بر مجموع امتیاز تمام اکشن های انتخاب نشده است) حالت دوم را انتخاب میکنیم.

سوال: در سناریوهای احتمالاتی که حرکات ارواح کاملاً قطعی 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.

میتوان میانگین وزن دار برگ های هر زیردرخت را محاسبه کرد (که وزن هر برگ احتمال رخ دادن آن میباشد.) و سپس بر اساس آن اکشنی که احتمال موفقیت بیشتر دارد را انتخاب کرد. در حالت غیر قطعی استفاده از minimax ممکن است باعث شود پکمن با در نظر گرفتن بدترین حالت، حتی در صورتی که احتمال وقوعش خیلی کم است، مدت طولانی وقت از دست بدهد و امتیاز منفی بگیرد. ولی در سناریو های قطعی باید آن حالت که قطعی که بدترین حالت است را در نظر بگیریم زیرا با استفاده از کمینه احتمال، احتمال باختن خیلی زیاد میشود. استراتژی بهینه پیچیدگی بیشتری دارد زیرا در سناریو قطعی نیاز نداریم تمام نود ها را گسترش دهیم و فقط بدتریم نود هر اکشن را گسترش میدهیم در حالی که در حالت غیر قطعی باید تمام برگ ها را داشته باشیم.

سوال: در سناریوهای احتمالاتی که عامل Pac-Man فقط مشاهدات جزئی از وضعیت بازی دارد، چگونه عامل می تواند از probabilistic minimax یا Expectimax احتمالی برای تصمیم گیری بهینه استفاده کند؟ چگونه عامل می تواند باور خود را در مورد وضعیت بازی بر اساس مشاهدات جدید به روز کند؟ مبادلات بین اکتشاف و بهره برداری را در سناریوهای مشاهده جزئی[4] تجزیه و تحلیل کنید و آنها را با مبادلات در سناریوهای مشاهده کامل مقایسه کنید. مدل های مختلف مشاهده جزئی و تأثیر آنها بر فرآیند تصمیم گیری را در نظر بگیرید.

عامل باید بتواند در طول زمان مشاهدات خود را حفظ کند و بر اساس تجربه گذشته و مشاهدات حال حاضر یک مدل و درخت تصمیم گیری مناسب تولید کند و بهترین انتخاب ممکن را انجام دهد.

اکتشاف و بهره برداری در حالت مشاهده جزیی بسیار اهمیت دارند و بر هم دیگر اثر میگذارند زیرا در یک زمان نمیتوانیم کل دنیا را مشاهده کنیم پس نیاز داریم از اکتشافات گذشته هم در کمار مشاهدات استفاده کنیم. ولی در حالت مشاهده کامل اهمیت اکتشاف خیلی کمتر است زیرا بر تمام اجزای بازی اشراف داریم و نیاز زیادی به اکتشاف نداریم.

به همین دلایل تصمیم گیری در حالت مشاهده جزیی بسیار سختتر و با خطای بیشتری میباشد و همچنین مدل سازی بسیار سخت تر است.

۵)

سوال: تفاوت‌های 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 تنها از استیت فعلی استفاده شده است و نیازی به اکشن نداریم.

تفاوت تابع ارزیابی در افزایش تاثیر روح های آسیب پذیر است که در اینجا طوری برنامه ریزی شده است که هنگامی که روحی آسیب پذیر پیدا میشود سعی میکند فاصله را با آن کم کند و با کشتن آن امتیاز بگیرد. در حالی که در بخش اول روح آسیب پذیر را صرفا در نظر نمیگرفت.

سوال: طراحی بهینه یک تابع ارزیابی برای یک عامل Pac-Man چیست؟ آیا می توان یک تابع ارزیابی طراحی کرد که بازی بهینه را در همه حالت های بازی تضمین کند؟ اگر نه، محدودیت های رویکرد عملکرد ارزیابی چیست؟ معاوضه بین دقت و کارایی محاسباتی° در طراحی تابع ارزیابی را تجزیه و تحلیل کنید. طراحی های مختلف عملکرد ارزیابی و تأثیر آنها بر رفتار عامل را در نظر بگیرید.

طراحی بهینه طراحی است که سعی کند با اکتشاف نود های کمتر و در زمان کمتری به جواب بهتر با امتیاز بیشتری برسد. خیر نمیتوان یک تابع ارزیابی داشت که در تمام حالات بازی بهینه بودن را تضمین کند. از محدودیت های آن میتوان به قابل پیش بینی نبودن حرکات فرد مقابل اشاره کرد که ممکن است سناریویی را پیش بگیرد که احتمال بهینه بودن جپاب ما را کاهش دهد. همچنین محدودیت های دیگری از جمله ناتوانی سیستم در در نظر گرفتن تمام حالات یک سناریو هم وجود دارد که باعث میشود نیاز به در نظر گرفتن حداکثر عمق شویم که میتواند بهینه بودن را کاهش دهد. به همین دلیل افزایش دقت میتواند کارایی محاسباتی را دچار مشکل کند زیرا ممکن است سیستم توانایی تجزیه و تحلیل نود های بسیار زیاد را نداشته 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 مقایسه‌تان از دقت، کارآیی محاسباتی و دیگر فاکتورهای لازم استفاده کنید.