علی بازشوشتری ۹۹۲۳۰۰۸

گزارش پروژه‌ی دوم مبانی و کاربردهای هوش مصنوعی

بخش 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 پکمن از دونه‌های غذا: مقادیر این پارامتر را به صورت معکوس به امتیاز اضافه میکنیم. این کار سبب میشود تا پکمن سعی کند خود را به دانه های غذا نزدیک تر کند و هرچه در مجموع فاصله اش کمتر باشد، امتیاز بیشتری کسب میکند.

اگر روح‌ها در حالت ترسیده باشند، فاصله پکمن از آنها را به امتیاز اضافه میکنیم.

اگر روح‌ها نترسیده باشند و فاصله منهتن پکمن از آنها بیش از ۱ باشد، به جهت دوری از روح‌ها، فاصله را به صورت معکوس از امتیاز کم میکنیم . اگر فاصله کمتر از ۱ بود، ۵۰۰ تا از امتیاز کم میکنیم تا از خورده شدن پکمن جلوگیری کنیم.

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)

راه های مختلفی برای این کار وجود دارد که به آنها در بخش قبل نیز اشاره شد. به عنوان مثال اضافه کردن فاصله از غذا به صورت معکوس یا دادن ضریب منفی به فاصله از روح ها. حتی برای حالات خاص نیز میتوان مقادیر را مشخص کرد مثلا هنگامی که روح در یک قدمی ما قرار دارد، مقدار زیادی از امتیاز پکمن را کم میکنیم.

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.

```
successorGameState = currentGameState.generatePacmanSuccessor(action)
newPos = successorGameState.getPacmanPosition()
newFood = successorGameState.getFood()
newGhostStates = successorGameState.getGhostStates()
newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
score = 0
food_dist = [util.manhattanDistance(newPos, food) for food in newFood.asList()]
if len(food_dist) > 0:
    score += 1.0 / min(food_dist)
else:
    score += 1
ghost_dist = [util.manhattanDistance(newPos, ghost.getPosition()) for ghost in newGhostStates]
for dist, time in zip(ghost_dist, newScaredTimes):
    if time > 0:
        score += dist
    else:
        if dist <= 1:
            score -= 500
        else:
            score -= 1 / dist
score += successorGameState.getScore()
return score
```

ابتدا فاصله منهتن تا غذاها را پیدا میکنیم و به صورت معکوس به امتیازها اضافه میکنیم.

پس از آن فاصله منهتن پکمن تا روح ها را محاسبه میکنیم. اگر روح ها در حالت ترسیده باشند، فاصله را به امتیاز اضافه میکنیم. سپس حالتی را بررسی میکنیم که روح‌ها نترسیده باشند. در این حالت اگر فاصله کمتر از ۱ باشد، امتیاز پکمن را به میزان ۵۰۰ تا کاهش میدهیم که نشان دهنده ی نامطلوب بودن وضعیت است و موجب میشود تا پکمن از روح فاصله بگیرد و توسط روح خورده نشود. اما اگر فاصله از روح بیشتر از ۱ باشد، معکوس فاصله را از امتیاز کم میکنیم. این مورد موجب میشود تا اولا پکمن مکان روح ها را در نظر بگیرد و سعی کند به سمت آنها نرود تا از خورده شدنش اجتناب شود و دوما در شرایط مساوی و برابر، انتخاب و تصمیمی را بگیرد که موجب دور شدنش از روح می‌شود.

بخش 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؟

با گذشت زمان و انجام حرکت از امتیاز کم می‌شود. به همین دلیل وقتی پکمن مطمئن میشود که در هر صورت میمیرد، بهترین امتیازی که میتواند کسب کند در حالتی است که هرچه زودتر خودکشی کند و ببازد تا امتیاز کمتری ازش کم شود.

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

```
pacmanIndex = 0
ghostIndices = range(1, gameState.getNumAgents())
legalActions = gameState.getLegalActions(pacmanIndex)
bestAction = ''
bestScore = float('-inf')
```

در ابتدا اندیس پکمن و روح ها را مشخص میکنیم. سپس کنش‌های مجاز را بدست آورده و همچنین مقدار اولیه امتیاز را برابر منفی بینهایت قرار میدهیم.

```python
def min_value(gameState, depth, ghostIndex):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    legalActions = gameState.getLegalActions(ghostIndex)
    score = float('inf')

    for action in legalActions:
        successor = gameState.generateSuccessor(ghostIndex, action)
        if ghostIndex == max(ghostIndices):
            score = min(score, max_value(successor, depth + 1))
        else:
            score = min(score, min_value(successor, depth, ghostIndex + 1))

    return score
```

```python
def max_value(gameState, depth):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    legalActions = gameState.getLegalActions(pacmanIndex)
    score = float("-inf")

    for action in legalActions:
        successor = gameState.generateSuccessor(pacmanIndex, action)
        score = max(score, min_value(successor, depth, ghostIndices[0]))

    return score
```

۲ تا تابع کمکی min_value و max_value را تعریف میکنیم.

در min_value باید مقدار مینیمم فرزندان گره انتخاب شود که برای روح‌ها باید از این تابع استفاده شود. برای روح ها کنش های مختلف مجاز در نظر گرفته میشوند و هرکدام به عنوان یک گره successor در نظر گرفته میشوند. اگر مقدار ghostIndex برابر با تعداد روح ها باشد یعنی همه ی روح ها نوبت خود را بازی کرده اند و اکنون نوبت عامل پکمن است پس به عمق بعدی در درخت رفته و تابع max_value را برای successor فراخوانی میکنیم(چون در حال

بدست آوردن مقدار گره برای پکمن هستیم که یک عامل maximizer است). درغیر اینصورت یعنی هنوز روح هایی هستند که نوبت خود را بازی نکرده اند پس در همان سطح باقی میمانیم و اندیس روح ها را افزایش میدهیم و به روح بعدی میرویم تا آن هم نوبت خود را بازی کند و مینیمم امتیازی که میتواند بدست آورد را برایش محاسبه میکنیم(به کمک همان تابع min_value زیرا پکمن یک عامل minimizer است)

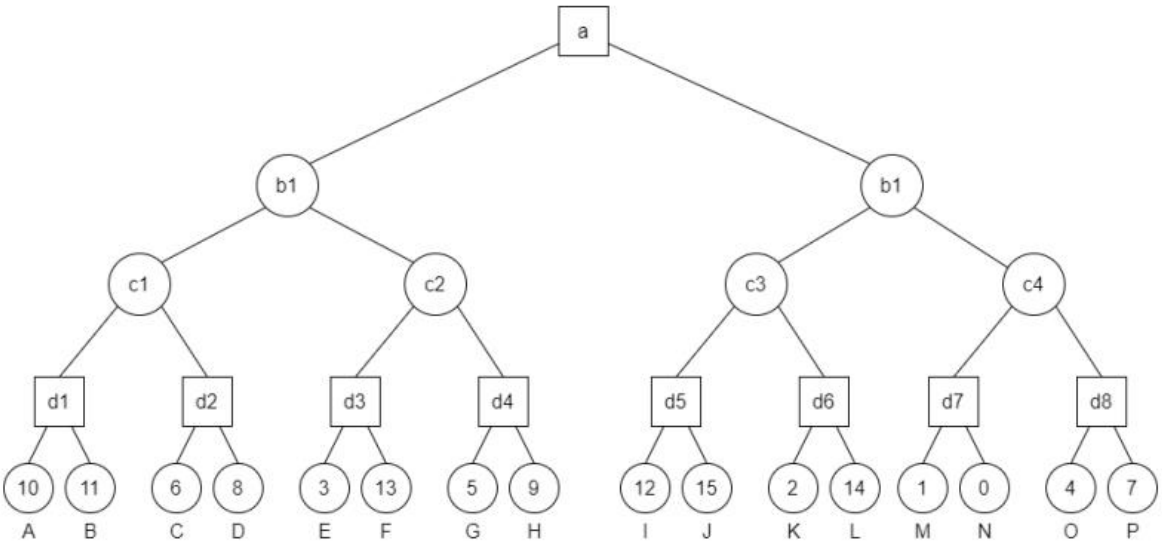
دقت شود که در هردو حالت مقدار مینیمم بین امتیاز فعلی و امتیاز برگردانده شده از تابع فراخوانی شده محاسبه میشود زیرا min_value برای روح ها بکار میروند و روح ها هم قصد کاهش امتیاز را دارند.

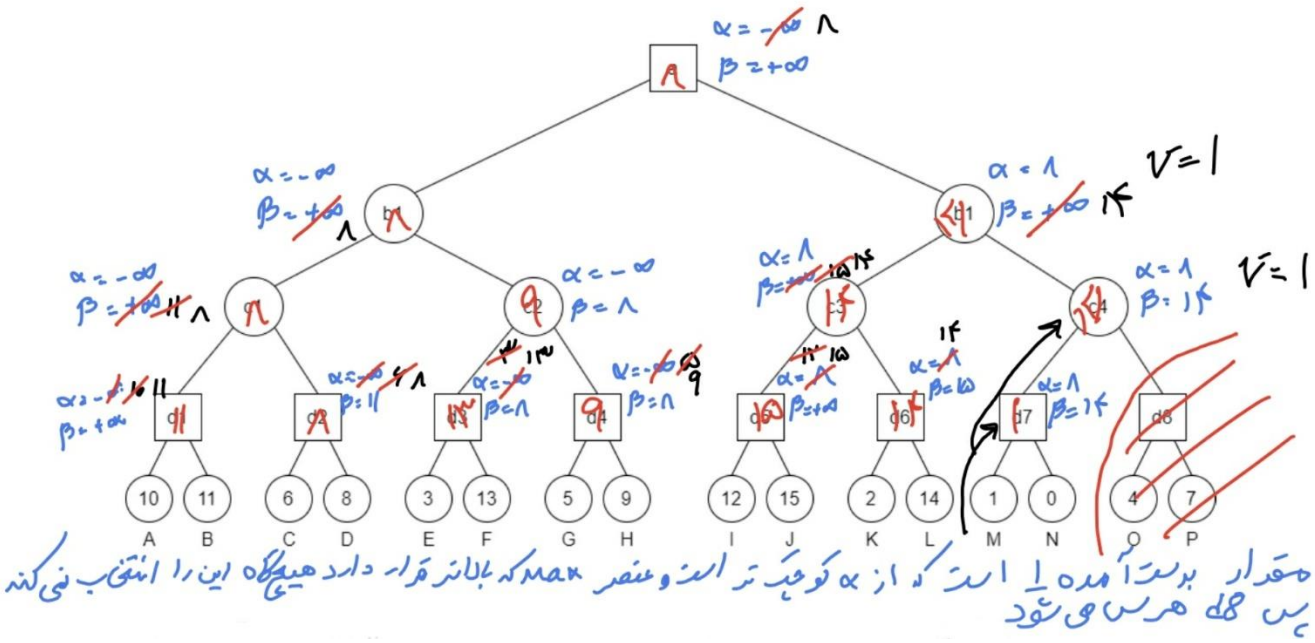
در max_value برخلاف min_value، مقدار ماکسیمم فرزندان گره باید انتخاب شود. برای هر کنش مجاز، یک گره ی successor در نظر گرفته میشود که این گره های فرزند مربوط روح ها هستن پس مقدار min_value خود را برمیگردانند. در نهایت پکمن(عامل maximizer) از بین گزینه‌های موجود و فرزندانش، راه و امتیاز ماکسیمم را انتخاب میکند

هرگاه به برگ ها که همان terminal stateها هستند و برد و باخت را مشخص میکنند برسیم، تابع evaluationFunction را صدا میزنیم.

بخش 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 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؟





D8 هرس میشود زیرا در هر صورت مقدار زیردرخت راست از زیردرخت چپ کمتر خواهد بود.

حرکت بعدی پکمن به سمت چپ خواهد بود.

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.

در حالت کلی و برای جواب نهایی خیر؛ مقداری متفاوت از حالت بدون هرس در ریشه تولید نمی شود زیرا ما تنها هنگامی اقدام به هرس کردن میکنیم که مطمئنیم بررسی بیشتر گره‌ها تغییری در جواب ایجاد نمی‌کند و گره ی پدر مقدارش را از فرزند دیگر انتخاب میکند.

Alpha: best already explored option along path to the root for maximizer

Beta: best already explored option along path to the root for maximizer

همانطور که از تعریف آلفا و بتا مشخص است، ما هرس را بر اساس ریشه انجام میدهیم. به همین علت تغییری در جواب بدست آمده برای ریشه به وجود نمی آید.

در گره‌های میانی ممکن است مقدار تغییر کند زیرا ما برخی از گره‌ها را هرس میکنیم و مقدارشان را بررسی نمیکنیم.(ممکن است در گره‌های بعدی مقداری بزرگ‌تر برای عنصر max و مقداری کوچک‌تر برای عنصر min پیدا شود. اما با اینحال این موارد تاثیری در جواب نهایی و ریشه ندارد چون میدانیم در سطح بالاتر مقدار این گره انتخاب نمیشود پس فرقی نمیکند مقدار آن برای عنصر max بزرگ تر باشد ویا برای عنصر min کوچکتر. به همین دلیل میتوانیم آنها را هرس کنیم.)

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

```
pacmanIndex = 0
ghostIndices = range(1, gameState.getNumAgents())
legalActions = gameState.getLegalActions(pacmanIndex)
bestAction = ''
bestScore = float('-inf')
alpha = float('-inf')
beta = float('inf')
```

کد مشابه قبل است و تنها مقادیر آلفا و بتا به آن اضافه شده است که بر اساس آن هرس انجام میشود.

ابتدا آلفا برابر منفی بینهایت و بتا برابر مثبت بی نهایت است

```python
def min_value(gameState, depth, ghostIndex, alpha, beta):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    legalActions = gameState.getLegalActions(ghostIndex)
    score = float('inf')

    for action in legalActions:
        successor = gameState.generateSuccessor(ghostIndex, action)
        if ghostIndex == max(ghostIndices):
            score = min(score, max_value(successor, depth + 1, alpha, beta))
        else:
            score = min(score, min_value(successor, depth, ghostIndex + 1, alpha, beta))

        if score < alpha:
            return score

        beta = min(beta, score)

    return score
```

همانطور که مشاهده می‌شود قسمت ابتدایی کد و تابع min_value مشابه قبل است که توضیحات آن را در بخش قبل دادیم. تنها تفاوت مربوط به انتهای کد است؛ جایی که امتیاز به دست آمده تاکنون را با مقدار آلفا مقایسه میکنیم. اگر امتیاز از آلفا کمتر باشد پس یعنی مسیر دیگر و بهتری برای عنصر maximizer در سطوح بالاتر وجود داشته ودر نتیجه عنصر maximizer آن راه بهتر را انتخاب میکند و سراغ شاخه ای که در حال حاضر در حال محاسبه ی گره های آن هستیم نمی آید. پس از تابع return میکنیم و دیگر مقدار باقی فرزندان را محاسبه نمیکنیم و هرسشان میکنیم.

اگر امتیاز از آلفا کمتر نباشد، بین امتیاز بدست آمده تاکنون و بتا، مینیمم میگیریم تا مینیمم امتیاز بدست آمده مسیر تا ریشه برای آن گره minimizer مشخص شود.

```python
def max_value(gameState, depth, alpha, beta):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    legalActions = gameState.getLegalActions(pacmanIndex)
    score = float('-inf')

    for action in legalActions:
        successor = gameState.generateSuccessor(pacmanIndex, action)
        score = max(score, min_value(successor, depth, ghostIndices[0], alpha, beta))

        if score > beta:
            return score

        alpha = max(alpha, score)

    return score
```

برای max_value هم به شکل مشابه و برعکس است. ابتدای کد مشابه کد موجود در بخش قبل و کلاس minimax است.

در انتها امتیاز بدست آمده تاکنون را با بتا مقایسه میکنیم و اگر بیشتر بود return میکنیم زیرا مسیر بهتر با امتیاز کمتری برای عنصر minimizer در سطوح بالاتر وجود داشته در نتیجه عنصر minimizer آن راه بهتر را انتخاب میکند و وارد شاخه ای که در حال محاسبه ی مقادیر آن هستیم نمیشود. پس دیگر لزومی ندارد که مقدار گره های بعدی و فرزندان را محاسبه کنیم و هرسشان میکنیم.

اگر امتیاز از بتا بیشتر نباشد، بین امتیاز بدست آمده تاکنون و آلفا، ماکسیمم میگیریم تا مکسیمم امتیاز بدست آمده مسیر تا ریشه برای آن گره maximizer مشخص شود.

بخش ۴)

توضیح کد:

```
def expected_value(gameState, depth, ghostIndex):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    legalActions = gameState.getLegalActions(ghostIndex)
    score = 0
    prob = 1.0 / len(legalActions)

    for action in legalActions:
        successor = gameState.generateSuccessor(ghostIndex, action)
        if ghostIndex == max(ghostIndices):
            score += prob * max_value(successor, depth + 1)
        else:
            score += prob * expected_value(successor, depth, ghostIndex + 1)

    return score

def max_value(gameState, depth):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    legalActions = gameState.getLegalActions(pacmanIndex)
    score = float("-inf")

    for action in legalActions:
        successor = gameState.generateSuccessor(pacmanIndex, action)
        score = max(score, expected_value(successor, depth, ghostIndices[0]))

    return score
```

همانطور که مشاهده می شود پیاده سازی مانند بخش های قبل است با این تفاوت که به جای min_value، expected_value را داریم که در آن یک احتمال یکسان برای هر action بدست آمده است(۱ تقسیم بر تعداد actionها) و سپس این احتمال در مقادیر ضرب شده و با امتیاز جمع بسته شده است و به عبارتی میانگین وزن دار گرفته شده است.

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
```

```
E:\AUT University & Academic Studies\Term 6\Artificial Intelligence\Project\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

E:\AUT University & Academic Studies\Term 6\Artificial Intelligence\Project\P2\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 died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Average Score: 15.0
Scores:        532.0, 532.0, -502.0, -502.0, -502.0, -502.0, 532.0, -502.0, 532.0, 532.0
Win Rate:      5/10 (0.50)
Record:        Win, Win, Loss, Loss, Loss, Loss, Win, Loss, Win, Win
```

در مینیماکس پکمن همواره میبازد اما در مینیماکس احتمالی یا همان expectimax پکمن برخی از دورهای بازی را برنده می شود.

در مینیماکس فرض ما بر این است که روح‌ها به بهینه ترین شکل ممکن بازی میکنند و بازی کاملا تخاصمی است (بدترین حالت ممکن را فرض می کنیم).

اما در expectimax ما با **عدم قطعیت** روبرو هستیم. این عدم قطعیت از دلایل مختلفی نظیر رندوم خارجی، احتمالی بودن محیط یا بهینه بازی نکردن عامل مقابل و **رندوم بازی کردن** آن وجود دارد. حتی یک حالت دیگر نیز آن است که ما میدانیم عامل مقابل چند استراتژی برای عمل

کردن دارد اما نمیدانیم که دقیقا کدام استراتژی را انتخاب میکند. بنابراین در expectimax برای گره های احتمالی، میانگین وزن دار از مقادیر فرزندان آن گره میگیریم.

حال در این مثال خاص، هنگامی که روح ها به صورت بهینه بازی میکنند (یا حداقل فرض ما این است) و پکمن میداند که قرار است ببازد، با رفتن به سمت روح ها و خودکشی کردن، امتیاز بهتری کسب میکند و از بیشتر کم شدن امتیاز اجتناب میکند.

اما هنگامی که روح ها به صورت بهینه بازی نمیکنند و تصادفی تصمیم می گیرند، احتمال این وجود دارد که روح ها به سمت پکمن نیایند و به جهت دیگری بروند و پکمن زنده بماند. پس در نتیجه ممکن است که پکمن صبر کند یا از روح ها دور شود بلکه زنده بماند و بتواند پیروز شود. به عبارتی در این حالت برخلاف قبل، پکمن شانسی را برای زنده ماندن و پیروز شدن خود نگه می دارد و این شانس را برابر ۰ نمی داند.

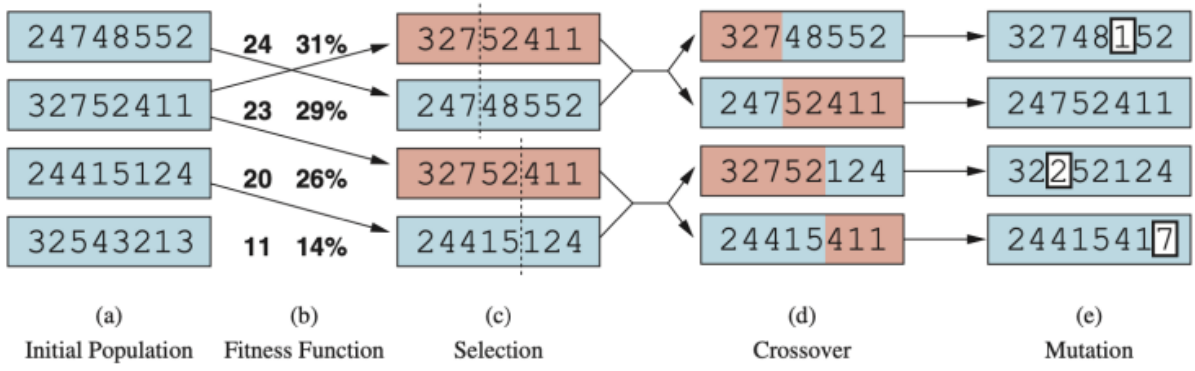
سوال: الگوریتم رولت ویل را بررسی کنید و بیان کنید که انتخاب هر کروموزوم در این الگوریتم بر چه اساسی است؟ اگر در بازی پکمن خودمان از آن استفاده کنیم، چه معیاری برای انتخاب هر **action** مناسب است؟ بر فرض اگر نیاز بود تا با کمک الگوریتم رولت ویل بیش‌تر از یک حالت انتخاب شود (با کمک مقدار تابع ارزیابی برای 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؟

هر کروموزم بسته به عملکردش در محیط بر اساس یک معیاری، یک مقدار فیتنس f_i دریافت می کند. سپس به هر کروموزم متناسب با عملکرد و مقدار فیتنسش، احتمالی برای انتخاب در نظر گرفته میشود. هرچقدر کروموزوم عملکرد بهتری داشته باشد و مقدار فیتنس آن بیشتر باشد، احتمال انتخاب آن کروموزوم بیشتر است. احتمال انتخاب کروموزم x به صورت زیر است.

$$p_x = \frac{f_x}{\sum_{i=1}^{n} f_i}$$

روش کار به این صورت است که یک عدد رندوم از ۰ تا جمع فیتنس ها تولید میشود. سپس فیتنس ها آنقدر با هم جمع میشوند تا جمعشان از آن عدد رندوم بیشتر شود. کروموزومی که فیتنس آن موجب exceed شدن کرده انتخاب میشود. این کار به تعداد کروموزوم های مورد نیاز انجام می شود. در مرحله ی بعد این کروموزوم ها با یکدیگر ترکیب می شوند تا کروموزوم های جدیدی تولید شوند.

در بازی پکمن می‌توان برای هر action یک عدد باینری صفر و یک در نظر گرفت که متناسب با آن یک حالت ایجاد میشود. به کمک این الگوریتم actionهارا ترکیب میکنیم تا به سود ماکسیمم برسیم. در آخر آن توالی actionهایی که منجر به رسیدن به حالت نهایی شده را انجام میدهیم.

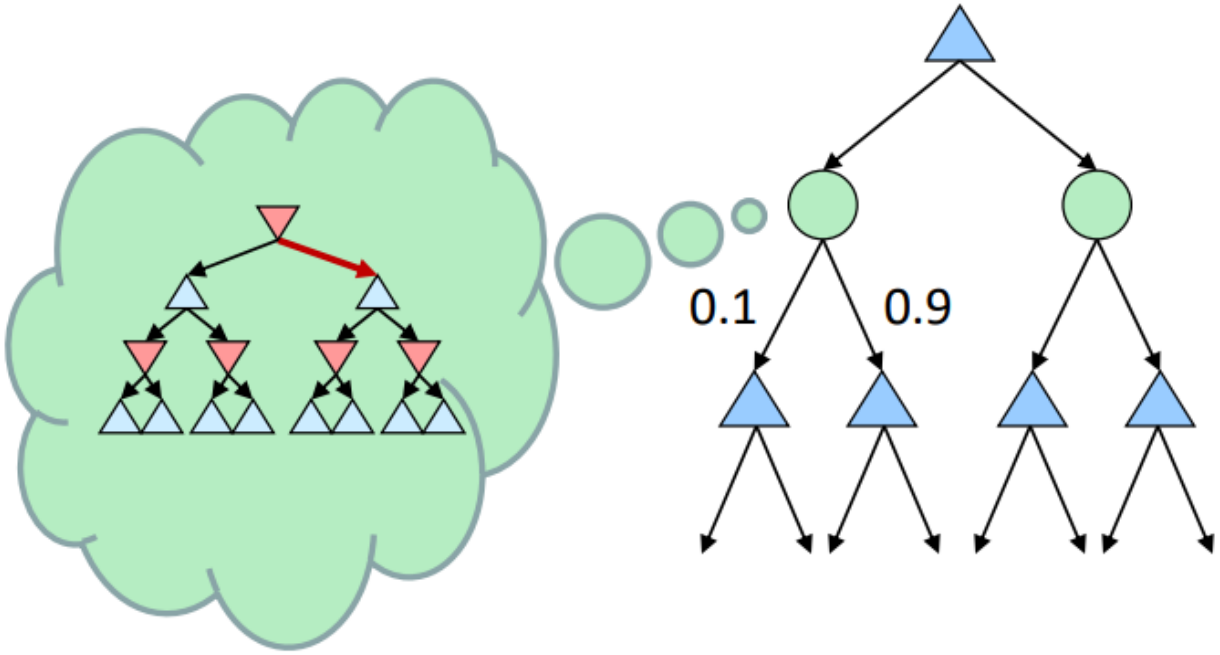


یا در یک حالت دیگر میتوان نسبت امتیاز هر action را به جمع امتیازها به عنوان احتمال درنظر گرفت. سپس مطابق الگوریتم رولت ویل، actionها را انتخاب کرده و کروموزوم های بعدی را میسازیم و در نهایت توالی حرکات تا رسیدن به حالت نهایی بدست می آید.

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.

الگوریتم expectimax. این الگوریتم به جای آنکه به طور قطعی مقدار min/max گره را حساب کند، برای گره هایی که حرکت انتخابی و بعدیشان به طور قطعی مشخص و معلوم نیست، میانگین وزن دار از فرزندانش می گیرد و به عنوان مقدار آن گره قرار میدهد. وقتی حرکت ارواح کاملا قطعی نیست، پس نمیتوان حرکت بعدی آنها را به قطع و یقین پیش بینی کرد بنابراین بهتر است میانگین وزن دار بگیریم تا متوجه شویم به طور نرمال، معمولا چه سناریویی و انجام چه حرکتی محتمل تر است. پس به طور خلاصه تفاوت این دو استراتژی در همان قطعی یا غیر قطعی بودن است که منجر به انجام محاسبات متفاوتی می شود.

پیچیدگی استراتژی بهینه بیشتر است. باید حرکت عامل مقابل را در چند حالت مختلف شبیه سازی و پیش بینی کنیم؛ و حتی در حالت بدتر ممکن است که لازم باشد عامل مقابل، حرکت ما را نیز پیش بینی کند! که خب سبب می شود پیچیدگی افزایش یابد و مدت زمان بیشتری جهت انجام محاسبات صرف شود.



همچنین مدل های احتمالی مختلف در محاسبات و تعداد شاخه ها نیز تاثیر دارند. اینکه در حالت غیرقطعی، برای هر گره غیرقطعی، چند استراتژی و راه مختلف وجود دارد. ۲ تا یا ۳تا یا ۱۰۰۰ تا!! و اینکه توزیع احتمال ها میان این استراتژی ها به چه شکل است و چگونه باید آنها را محاسبه کنیم که خود این نحوه توزیع احتمال در پیچیدگی و مدت زمان محاسبه تاثیر دارد.

اما در سناریوهای قطعی مثل مینیماکس، کلا یک درخت دارم و propertyها مشخص هستند. دیگر نیازی به انجام محاسبات اضافه نیست.

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[۴] 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.

هنگامی که عامل از مشاهده ی جزئی استفاده میکند، بهتر است که تجربیات قبلی خود را حفظ و ذخیره کند (در قالب internal state میتواند انجام شود) تا در ادامه تصمیمات بهتری اتخاذ کند. میتوان از الگوریتم های مبتنی بر Markov و Bayesian network جهت تصحیح احتمالات استفاده کرد.

در ارتباط با اکتشاف و بهره برداری: ممکن است عامل بخشی از زمان خود را صرف اکتشاف و شناسایی محیط و دیگر عامل کند تا در ادامه بتواند از اطلاعاتی که کسب کرده بهره برداری کند و احتمالاتی که در نظر میگیرد را اصلاح کند. البته بهتر است که این اکتشاف و بهره برداری به صورت همزمان انجام شود.

اکتشاف و بهره برداری در سناریوهای مشاهده جزئی بسیار مهم است زیرا اطلاعات ما نسبت به محیط ناقص و محدود است و طبعا چون اطلاعات کمتری داریم پس دقت ما نیز کمتر خواهد بود پس تصمیمات بدتری اتخاذ خواهیم کرد. در سناریوهای مشاهده کامل، اکتشاف و بهره برداری تقریبا معنایی ندارد زیرا ما به اطلاعات اشراف داریم و در حالات احتمالی، تنها نمیدانیم که کدام حالت و اتفاق به طور قطع میفتد که خب این احتمالی بودن جزوی از ویژگی محیط است و کاری در ارتباط با آن نمیتوان انجام داد. اما همینکه ما احتمالات را به طور دقیق و کامل میدانیم خودش بسیار مفید و کارامد خواهد بود.

بنا به دلایلی که گفته شد مدل سازی و تصمیم گیری در مشاهده جزئی سخت تر و پیچیده تر است و با دقت کمتری همراه است.

بخش 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.

```
pacmanPosition = currentGameState.getPacmanPosition()
foods = currentGameState.getFood()
ghostStates = currentGameState.getGhostStates()
scaredTimers = [ghostState.scaredTimer for ghostState in ghostStates]
ghostPositions = currentGameState.getGhostPositions()

food_weight = 10
ghost_weight = -1000
scared_ghost_weight = 200

score = currentGameState.getScore()

# Evaluate food
food_list = foods.asList()
if food_list:
    closest_food_dist = min([util.manhattanDistance(pacmanPosition, food) for
food in food_list])
    if closest_food_dist == 0:
        score += 1000  # Reward for eating food
    else:
        score += food_weight / closest_food_dist

# Evaluate ghosts
for ghostState, scaredTimer, ghostPosition in zip(ghostStates, scaredTimers,
ghostPositions):
    ghost_dist = util.manhattanDistance(pacmanPosition, ghostPosition)
    if scaredTimer > 0:
        score += scared_ghost_weight / (ghost_dist + 1)
    else:
        if ghost_dist <= 1:
            score += ghost_weight - 500  # Penalize for being too close to a
ghost
        else:
            score += ghost_weight / ghost_dist

return score
```

اولا در این کد اطلاعات از currentGameState استخراج میشوند. برخلاف قبل که از successorGameState و action استفاده میشد.

در این تابع ارزیابی، فاصله از نزدیک ترین دونه غذا را حساب کرده ایم که موجب می شود پکمن ترغیب شود به سمت آن حرکت کند. این کار را به این صورت انجام داده ایم که وزن غذا رالاوزنی

که خودمان تعیین کرده ایم) برا این فاصله تقسیم میکنیم. تقریبا مشابه قبل. هرچه فاصله نزدیک تر باشد، امتیاز بیشتری اضافه میشود.

همچنین به ازای خوردن هر غذا، به عامل امتیاز اضافه میکنیم.

معیار بعدی در این تابع، فاصله از روح است. اگر روح در حالت ترسیده باشد، روح را عملا مانند غذا میبینیم. تنها با این تفاوت که وزنی که برای آن در نظر گرفته ایم بیشتر است و موجب افزایش بیشتر امتیاز میشود. پس برای روح ترسیده، مشابه توضیحاتی که برای غذا دادیم(فاصله از آن و تقسیم وزن روح ترسیده بر فاصله) به امتیاز پکمن اضافه میکنیم.

اگر روح در حالت ترسیده نباشد و در یک قدمی پکمن باشد، امتیاز فراوانی را از پکمن کم میکنیم که سبب میشود از روح دوری کنیم و سعی کنیم که پکمن خورده نشود.

اما اگر روح در یک قدمی ما نباشد، امتیاز پکمن را به میزان تقسیم وزن روح بر فاصله از روح کم میکنیم.

همانطور که معلوم است در هنگام انجام محاسبات برای اثرگذاری روح ها، امتیاز به گونه ای تغییر میکند تا پکمن به سمت روح نرود.

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 در زمان کمتر با expand کردن گره‌های کمتر و همچنین انجام محاسبات کمتر(پیچیدگی محاسباتی) به جواب بهتر و با امتیاز بیشتر دست یابد. تابع ارزیابی هرچه دقیق تر با استفاده از اطلاعات موجود بهترین عمل را تشخیص دهد، بهینه تر است. خیر نمیتوان تابع ارزیابی ای را طراحی کرد که در همه حالات بهینه باشد زیرا ما میان دقت و

پیچیدگی محاسباتی trade off داریم. مسلما برای آنکه بتوانیم دقت را ببریم بالا، باید محاسبات بیشتری را انجام دهیم و فاکتورهای بیشتری را در محاسبات خود دخیل کنیم.

همچنین نحوه ی عملکرد عامل مقابل نیز تاثیرگذار است. اینکه به صورت بهینه بازی میکند یا به صورت احتمالی. به طور کلی محیط در عملکرد تاثیر گذار است.

همچنین باید محدودیت زمانی ویا محدودیت های سخت افزاری را نیز در نظر گرفت.

همانطور که گفته شد میزان محاسبات و دقت و فاکتورهای دخیل در طراحی عملکرد ارزیابی تاثیر گذار است. هر جنبه ای تاثیر متفاوتی بر عملکرد عامل میگذارد مانند خوردن غذا یا برخورد با روح.

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.