دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

درس هوش مصنوعی و کارگاه

گزارش ۳ : پیاده‌سازی الگوریتم Minimax برای بازی Pacman

نگارش

محمدامین عسکری

۴۰۰۱۲۰۳۳

۱۴۰۲/۰۸/۲۶

استاد

مهدی قطعی

استادیار

بهنام یوسفی

چکیده

در این مقال به گزارش پیاده‌سازی الگوریتم Minimax کلاسیک در محیط بازی پکمن پرداخته‌ایم. به دلیل ممکن نبودن پیمایش کامل درخت حالات، از جستجوی عمق‌محدود مینیمیکس(depth-limited minimax) استفاده شده است که مستلزم تعریف تابع ارزیابی بهینه‌ای برای این بازی بود. معیارهای انتخاب تابع ارزیابی و نتایج تست الگوریتم با این تابع گزارش شده‌اند. نهایتاً نسخه‌های دیگر این الگوریتم معرفی و نتایج پیاده‌سازی آن‌ها گزارش شده‌اند.

فهرست مطالب

صفحه

فصل نخست معرفی بازی پکمن و نحوه پیاده‌سازی آن

در این بازی، پکمن توسط بازی‌کننده در یک صفحه بازی هدایت می‌شود تا موانعی که جلوی او را می‌گیرند و روح‌هایی که می‌توانند او را نابود کنند پشت سرگذاشته و کلیه دانه‌های غذای موجود در صفحه را بخورد. اگر روحی پکمن را بگیرد باخته است اما پک‌من می‌تواند با خوردن یک کپسول، روح‌ها را بترساند که باعث می‌شود نتوانند تا مدتی آسیبی به او برسانند. مضافاً روح‌های ترسیده خوردنی‌اند و در صورت خورده شدن امتیازی بالا به پکمن می‌دهند.

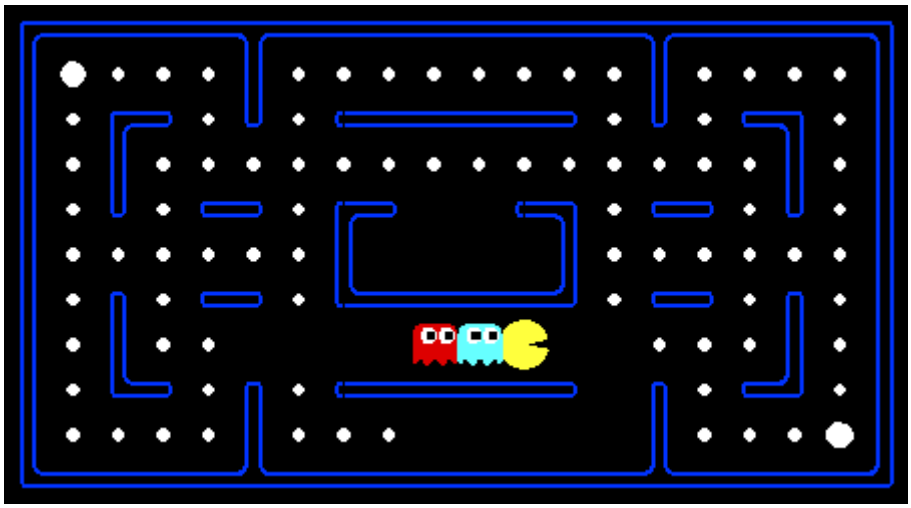
این بازی محیطی نسبتاً پیچیده دارد که برای پیاده‌سازی الگوریتم‌های مختلف هوش مصنوعی بسیار مناسب است.

برای تست الگوریتم مینمیکس در این بازی، از User Interface طراحی شده توسط دانشگاه برکلی به همین منظور استفاده شده است.

این UI از لینک زیر قابل دسترسی است:

https://inst.eecs.berkeley.edu/~cs188/fa23/projects/proj2/

تصویر محیط بازی و شکل پکمن و روح‌ها در این بازی به صورت زیر است:



این UI در فایل ghostAgents.py دو کلاس برای دو بازی مختلف برای روح‌ها تعریف کرده است. حالت دیفالت، کلاس RandomGhost است که نوع بازی آن از اسمش پیداست و حالت دیگر DirectionalGhost است که دائماً به سمت پکمن حرکت می‌کند و سعی می‌کند آن را بگیرد.

استراتژی‌های مختلف بازی پکمن، در فایل multiAgents.py در کلاس‌هایی مختلف تعریف می‌شوند. مثلاً کلاس MinimaxAgent از الگوریتم مینیمکس کلاسیک استفاده می‌کند تا در یک وضعیت (gameState) حرکت (action) مناسب را توسط تابع getAction خود برگرداند.

کلاس‌های دیگری مثل AlphaBetaAgent و ExpectimaxAgent نیز تعریف شده‌اند که علاوه بر مینمیکس کلاسیک، آن‌ها هم پیاده‌سازی شده‌اند.

برای این که پکمن مثلاً با استراتژی مینیمکس بازی کند، کافی است در getAction حرکتی که با مینیمکس با عمق دلخواه بدست می‌آید را با استفاده از متدهای کمکی و کلاس‌های از پیش تعریف شده حساب کنیم و نیازی به پیاده‌سازی چیز دیگری نیست.

تابع ارزیابی دیفالتی بازی تعریف شده است. در این تابع ارزیاب، به ازای خوردن هر واحد غذا ۱۰ امتیاز به پکمن داده می‌شود. خوردن روح‌های ترسیده ۲۰۰ امتیاز می‌دهد و بردن بازی ۵۰۰ امتیاز می‌دهد و بازی تمام می‌شود. خورده شدن توسط روح با کسر ۵۰۰ امتیاز بازی را خاتمه می‌دهد. خوردن کپسول‌ها مستقیماً امتیازی اضافه نمی‌کند. علاوه بر پیاده‌سازی getAction کلاس‌های مختلف multiagent یک تابع ارزیاب بهتر هم تعریف می‌کنیم .

این تابع ارزیاب در متد betterEvaluationFunction قابل دسترسی است.

در انتهای این بخش نحوه تست الگوریتم‌های مختلف بازی پکمن و تابع ارزیاب پیاده‌سازی شده با عمق دلخواه و نتایج آن به تفصیل توضیح داده شده است. در ادامه مروری کوتاه و مختصر به تئوری الگوریتم‌های پیاده‌سازی شده برای پکمن و نحوه کار تابع ارزیاب پیاده‌سازی شده خواهیم داشت.

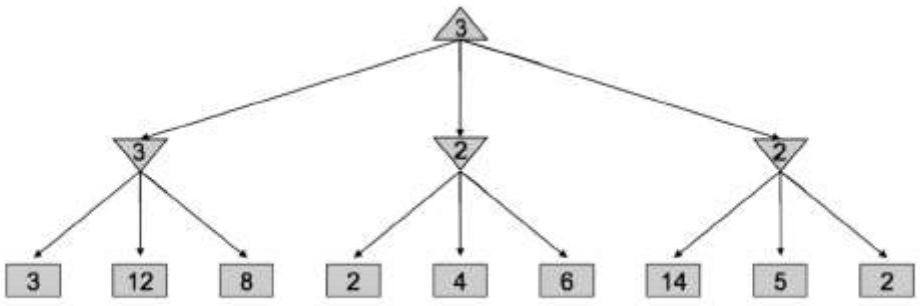
فصل دوم

معرفی الگوریتم مینیمکس و نسخه‌های مختلف آن

2-1- شرح الگوریتم مینیمکس کلاسیک، مینیمکس عمق محدود و تابع ارزیاب

الگوریتم مینیمکس کلاسیک برای بازی‌های zero-sum طراحی شده است. در این بازی‌ها، وضعیت برد برد وجود ندارد و کسب امتیاز برای هر بازیکن به منزله کسر امتیاز از بازیکن دیگر است. این یعنی در حالتی که دو بازیکن داریم، امتیاز(یا utility) بازیکن دوم را منفی امتیاز بازیکن اول در نظر بگیریم یا اینکه هدف بازیکن دوم را بجای ماکسیمم کردن امتیاز خودش، بصورت مینیمم کردن امتیاز بازیکن اول مدل کنیم. در حالت دوم، با فرض اینکه بازیکن‌ها به نتایج همه حرکت‌ها آگاهند و بهترین حرکت را انتخاب می‌کنند می‌توانیم فرض کنیم بازیکن اول در نوبت خودش حرکتی را انجام می‌دهد که به بیشترین utility یا امتیاز منتهی می‌شود و بازیکن دوم حرکتی را انجام می‌دهد که به کمترین امتیاز منتهی می‌شود این یعنی انتخاب هردو بازیکن به انتخاب بازیکن دیگر بستگی دارد.

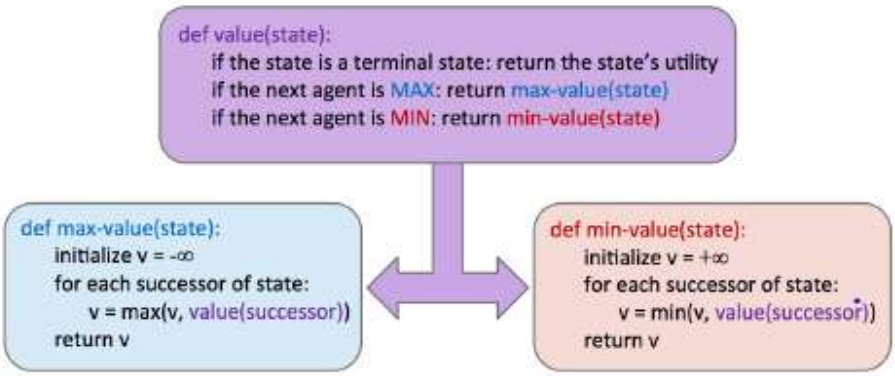
برای مثال بازی‌ای را در نظر بگیرید که در دو حرکت تمام می‌شود و هر بازیکن باید بین ۳ حرکت انتخاب کند. مطابق شکل زیر نظیر هر زوج حرکت بازیکن‌ها می‌توان یک درخت ترسیم کرد که گره‌های آن نشان دهنده حالت‌های بازی پیش از پایان و برگ‌های آن نشان‌دهنده امتیاز نهایی در هنگام پایان بازی هستند.



بازیکن اول با یک حرکت به یکی از سه گره میانی می رود، اما می‌دلند بازیکن دوم در هر حالت از بین برگ‌ها(یا حرکتهای منتهی به برگها) آنی را انتخاب می‌کند که امتیاز را کمتر می‌کند پس اگر گره اول از چپ را بردارد، امتیازش ۳ خواهد بود. اگر گره دوم یا سوم را بردارد، امتیازش ۲ خواهد بود. پس گره اول از چپ را انتخاب می‌کند و بازیکن دوم از بین برگ‌ها، مجبور است برگ اول از چپ را بردارد، چون در غیر این صورت امتیاز بیشتری را به او واگذار می‌کند و بازی با امتیاز ۳ خاتمه خواهد یافت.

به همین ترتیب در درختی بزرگتر متعلق به بازی‌ای با حرکات و نوبت‌های بازی بیشـتر می‌توانیم بهترین حرکت و امتیاز نهایی برای هر بازیکن را به شرط بهینه بازیکن دیگر بازکن، در هر گره حساب کنیم.

شبه‌کد این الگوریتم که موسوم به مینیمکس می‌باشد اینگونه است:



در حالتی که بیش از دو حریف وجود دارد، مثل پکمن با دو روح، برای هر دو حریف باید تابع min را فراخوانی کنیم.

بدیهی است که این الگوریتم باید همه حرکت‌های ممکن را تا رسیدن به وضعیت برد یا باخت دنبال کند که در بازی‌ای مثل پکمن با ضـریب انشـعاب حدود ۱۶ و عمق جسـتجویی حداقل به تعداد غذاها که می‌تواند عددی قریب به ۸۰ در یک صفحه متوسط باشد، محاسبه کلیه برگ‌های درخت و بازگشت روی کلیه گره‌های میانی غیرممکن است.

به همین دلیل عمق جسـتجو را محدود نگه می‌داریم و گره‌های پایانی را برگ می‌گیریم.به این برگ‌ها که هر یک نماینده یک وضـعیت غیرپایانی بازی هسـتند با تابع ارزیاب، امتیازی نسـبت می‌دهیم و با آن حرکت بهینه را حساب می‌کنیم. امتیاز هر وضعیت باید امکان مقایسه بین حرکت‌ها و وضعیت‌ها را بدهد تا مناسب باشد. مثلاً حالتی که غذاهای بیشتری خورده شده‌اند باید امتیاز بیشتری داشته باشد، همینطور حالتی که پکمن به غذا نزدیک‌تر است. به عنوان مثالی دیگر،حالتی که در آن روح‌ها بیش از حد به پکمن نزدیک می‌شـوند نامناسـب اسـت اما اگر روح‌ها ترسـیده باشـند پکمن باید به آن‌ها نزدیک شـود تا بتواند بخوردشان پس حالت نزدیکتر امتیاز بهتری دارد.

در ادامه کدی که برای مینیمکس زده‌ایم را می‌بینید. تابع minimax معادل value در شکل بالا است با این تفاوت که اولاً علاوه بر امتیاز حرکت را هم بر می‌گرداند و به عمق جستجو (که دیفالت آن ۲ است ولی همچنان که در فصل ۳ می‌بینیم قابل تغییر می‌باشد) هم توجه می‌کند و امکان بازی با چند روح را می‌دهد. تابع‌های max_move و min_move هم مشابه‌اند. در انتها می‌بینید که مینیمکس برای پکمن فراخوانی شده است و حرکتی که بر می‌گرداند در getAction سپس return شده است.

```
def miniMax(state, agent, depth):
    agent = agent % gameState.getNumAgents()
    if isTerminal(state):
        return self.evaluationFunction(state), 'Stop'
    if isPacman(agent):
        if depth == self.depth:
            return self.evaluationFunction(state), 'Stop'
        return max_move(state, agent, depth + 1)
    else:
        return min_move(state, agent, depth)

def max_move(state, agent, depth):
    legalMoves = state.getLegalActions(agent)
    bestUtil = float('-inf')
    bestMove = 'Stop'
    for move in legalMoves:
        stateAfterMove = state.generateSuccessor(agent, move)
        stateUtil, nextAgentMove = miniMax(stateAfterMove, agent + 1,
depth)
        if stateUtil > bestUtil:
            bestUtil = stateUtil
            bestMove = move
    # if depth == 1:
    #     print("best move and util at depth 1 == ",bestUtil, bestMove)
    return bestUtil, bestMove

def min_move(state, agent, depth):
    legalMoves = state.getLegalActions(agent)
    bestUtil = float('inf')
    bestMove = 'Stop'
    for move in legalMoves:
        stateAfterMove = state.generateSuccessor(agent, move)
        stateUtil, nextAgentMove = miniMax(stateAfterMove, agent + 1,
depth)
        if stateUtil < bestUtil:
            bestUtil = stateUtil
            bestMove = move

    return bestUtil, bestMove

minimaxUtil, minimaxMove = miniMax(gameState, 0, 0)
return minimaxMove
```

در ادامه به بررسی تابع ارزیاب خواهیم پرداخت. این تابع را ذیلاً می‌بینید.

```
def betterEvaluationFunction(currentGameState):
    if currentGameState.isWin():
        return 100000000 + currentGameState.getScore()
    elif currentGameState.isLose():
        return -100000000 + currentGameState.getScore()

    currentPacmanPos = currentGameState.getPacmanPosition()
    currentFoodPositions = currentGameState.getFood().asList()
    currentGhostStates = currentGameState.getGhostStates()
    # currentGhostPositions = [ghostState.getPosition() for ghostState
in currentGhostStates]
    currentScaredTimes = [ghostState.scaredTimer for ghostState in
currentGhostStates]
    currentCapsules = currentGameState.getCapsules()

    distanceFromPacman = lambda pos:
util.manhattanDistance(currentPacmanPos, pos)
    addedScore = 0

    addedScore += 20 * 1 / nearestFoodsBFS(currentGameState,
currentFoodPositions, 3)  # 20
    addedScore -= 30 * len(currentFoodPositions)

    # currentGhostStates = sorted(currentGhostStates, key=lambda
ghostState: ghostState.scaredTimer, reverse=True)

    for ghostState in currentGhostStates:
        ghostDistance = distanceFromPacman(ghostState.getPosition())

        ghostScaredTimeLeft = ghostState.scaredTimer

        if ghostScaredTimeLeft < ghostDistance:
            if ghostDistance <= 4:
                addedScore -= 100
            elif ghostDistance <= 3:
                addedScore -= 5000
                if ghostDistance <= 2:
                    addedScore -= 200000
            # else:
            #     addedScore -= 1 / ghostDistance
        else:
            addedScore += 3 / ghostDistance #10*ghostScaredTimeLeft

    minScaredTimeLeft = min(currentScaredTimes)

    if currentCapsules:
        addedScore += sum([1 / distanceFromPacman(capsule)
                           for capsule in currentCapsules
                           if minScaredTimeLeft <= 0])

    return addedScore + 0.2 * currentGameState.getScore()
```

در این تابع ابتدا موقعیت پکمن، غذاها، روح‌ها، کپسول‌ها و تایمر ترسیدگی روح‌ها از توابع کمکی گرفته می‌شود.

در ادامه یک امتیاز اضافی(added score) محاسبه می‌شود که با score تابع ارزیاب دیفالت UI با ضریب 0.2 ترکیب می‌شود. امتیاز اضافی سعی می‌کند وضعیتهای بهتر صفحه را امتیاز بهتر بدهد.

مواد سازنده امتیاز، مولفه‌هایی مختلف اند که در ادامه شرح داده‌ایم. ضرایب و فرمت این موارد کمی رندم است و با آزمون و خطا بدست آمده است.

مواد سازنده امتیاز اضافی اینهایند:

1)فاصله تا نزدیک ترین غذا یا مجموع فاصله تا نزدیک‌ترین غذاها: این عدد هرچه کمتر بهتر، یعنی عکس این عدد هرچه بیشتر باشد بهتر است. معکوس کردن این امکان را می‌دهد که رنج عدد را کنترل کنیم و این برای ترکیب مولفه‌های با رنج‌های طبیعی مختلف مثل فاصله با غذا و و فاصله تا روح و کپسول و ... بسیار مفید است چون امکان تنظیم تاثیر هر مولفه در مجموع را می‌دهد. معکوس را با ضریب ۲۰ وارد معادله می‌کنیم.
می‌توانیم با مجموع فاصله تا ۳ یا ۵ غذای نزدیکتر کار کنیم یا مجموع فواصل همه غذاها یا صرفاً فاصله تا نزدیکترین غذا. اولی زیادی حریصانه است و یک غذای نزدیک را به تعداد بیشتری غذا که کمی دورتر است ولی در جهت دقیقاً مخالف ترجیح می‌دهد. در مقابل کار کردن با فواصل همه غذاها سخت‌تر است چون ممکن است در جهتهای بسیار متفاوت باشند و جمع آنها بدون در نظر گرفتن جهتشان بی‌معناست و جمع آن‌ها با در نظر گرفتن جهتشان سخت است. اما وقتی تعداد غذاهای مدنظر را به ۵ یا حتی کمتر ۳ یا ۲ محدود می‌کنیم، احتمال این که غذاهای نزدیک کنار هم باشند بیشتر است پس جمع آن‌ها معنی‌دارتر می‌شود.

2) فاصله تا روح: اگر روح ترسیده باشد باید آن را بخوریم بهتر است، پس معکوس فاصله تا روح را را هم اگر تا رسیدن به روح همچنان ترسیده باشد با ضریب ۳ وارد می‌کنیم. در حالت عادی فاصله با روح را را کم نمی‌کنیم چون در آزمایشهای متعدد دیده شده باعث ترسو شدن پکمن می‌شود اما اگر روح بیش از حد به پکمن نزدیک بود، امتیاز زیادی کم می‌کنیم.

3) تعداد غذاهای مانده: هرچه کمتر بهتر پس با ضریب بزرگ ۳۰ از امتیاز کمش می‌کنیم.

4) کپسول‌ها: اگر هیچ روحی ترسیده نبود، در آن صورت عکس فاصله از آن با امتیاز پکمن جمع می‌شود.

قابل توجه است که در محاسبه فاصله روح‌ها و کپسولها از فاصله منهتن و در محاسبه فاصله غذا از فاصله با BFS که دقیق است استفاده کرده‌ایم. زیرا برای روح‌ها و کپسول‌ها فاصله دقیق روح آنقدر مهم نبود و حدس زده شد عدم استفاده از BFS زمان محاسبه را سریعتر می‌کند. به خصوص که برای غذا داریم فاصله دقیق را حساب می‌کنیم و همین کافیست تا هیچ وقت پکمن گیر نکند.

برای عمق جستجو از عمق دیفالت دو استفاده کرده‌ایم اما می‌توان این عمق را در کانستراکتور MultiAgentSearchAgent تغییر داد یا هنگام اجرا با دستور a depth=x- آنرا تغییر داد.

برای اجرای پکمن با استراتژیهای مختلف از دستورات زیر می‌توان استفاده کرد:

```
python pacman.py -p MinimaxAgent -a evalFn=better,depth=3
python pacman.py -p AlphabetaAgent -a evalFn=better,depth=2
python pacman.py -p ExpectimaxAgent -a evalFn=better
```

همچنین دستوری به شکل زیر نیز قابل اجرا است:

```
python pacman.py -p MinimaxAgent -q -n 10 -a evalFn=better
```

با دستور بالا، بدون گرافیک چندین بار مینیکمس با تابع ارزیابی ما ران می‌شود و میانگین امتیازات نمایش داده می‌شود.

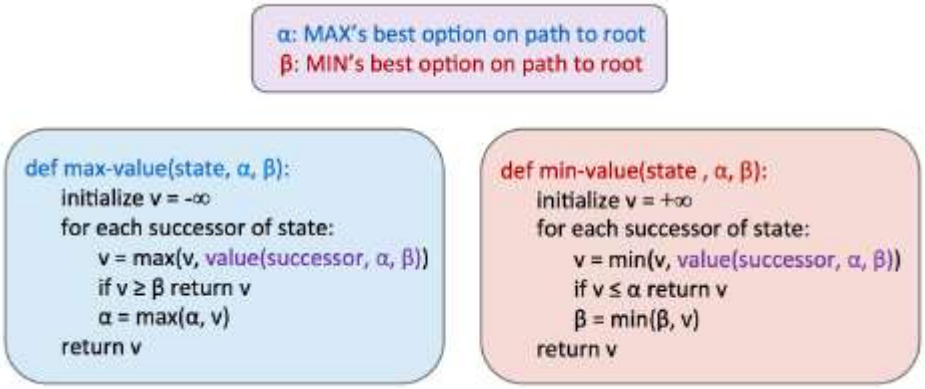
می‌توان نمره کلاسهای مختلف تعریف شده را از دستور زیر گرفت:

```
python autograder.py
```

با اضافه کردن q q2- می‌توان مینیمکس را تست کرد و عدد ۳و۴ به ترتیب آلفابتا و اکسپکتیمکس را و نهایتاً عدد ۵ خود تابع ارزیابی را تست می‌کند.

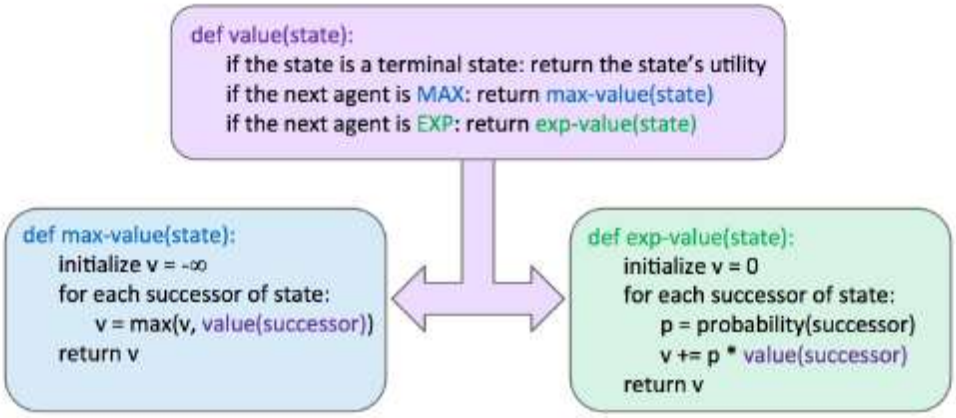
2-2- جستجو با هرس‌کردن آلفابتا و جستجوی تصادفی یا اکسپکتیمکس

الگوریتم هرس آلفابتا طبق شکل زیر پیاده‌شده است:



این الگوریتم از boundهایی که روی مقدار ماکس و مین وجود دارد برای هرس زیردرختهایشان استفاده می‌کند.

الگوریتم Expectimax بجای تابع مین، از امید ریاضـــی امتیاز بعد از حرکتهای مختلف روحها برای تصمیم گرفتن راجب حرکت استفاده می‌کند.



پیاده‌سازی این الگوریتم‌ها مشابهاً ذیل کلاس‌های هم‌نام انجام شده است.

فصل سوم

خاتمه و نتیجه‌گیری

پیاده‌سازی تابع ارزیابی مناسب برای الگوریتم مینیمکس بسیار مهم است زیرا در عمل عمق محدودی را می‌توانیم تست کنیم.

منابع و مراجع

[1] https://inst.eecs.berkeley.edu/~cs188/fa23/projects/proj2/

[2] https://inst.eecs.berkeley.edu/~cs188/fa18/
Lecture : Game Trees: Minimax and Game Trees: Expectimax, Utilities

Abstract

In this report, different algorithms of A* algorithm family are explained and compared.

Amirkabir University of Technology
(Tehran Polytechnic)

Mathematics and Computer Science Department

Report 1: An investigation of Rational Agents in three domains

By
Mohammad Amin Askari

Professor
Dr. Mahdi Ghatiee

Teacher Assistant
Dr. Behnam Yousefi

Month & Year:
2023/10/30