دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

درس مبانی و کاربردهای هوش مصنوعی

گزارش پروژه سوم (Connect Four Game)

نام و نام‌خانوادگی : مهدی رهبر

شماره دانشجویی : ۴۰۱۳۶۱۳۰۳۶

زمستان ۱۴۰۳

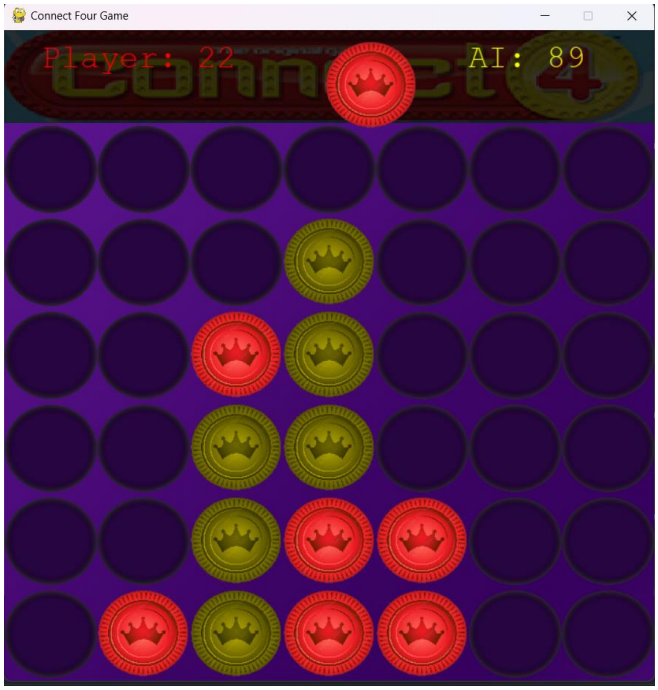
- **الگوریتم Minimax**

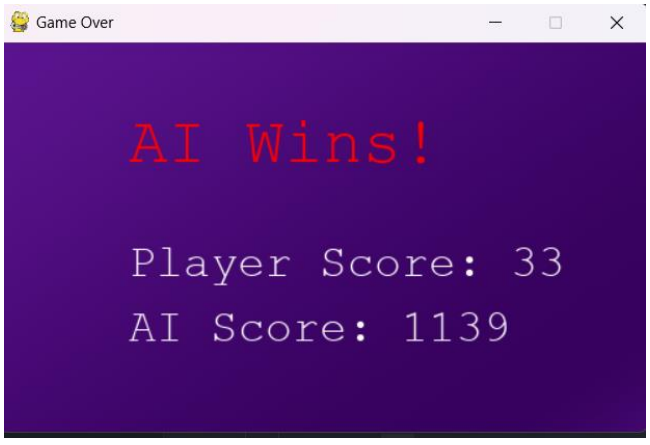
الگوریتم **Minimax** یک روش بازگشتی در تئوری بازی‌ها و تصمیم‌گیری است که برای پیدا کردن بهترین حرکت در یک بازی دو نفره با مجموع صفر استفاده می‌شود. این الگوریتم با شبیه‌سازی تمام حرکات ممکن (درخت بازی) کار می‌کند و در هر مرحله دو بازیکن در نظر گرفته می‌شوند: **Maximizer** (بازیکنی که سعی در بیشینه کردن امتیاز دارد) و **Minimizer** (بازیکنی که سعی در کمینه کردن امتیاز Maximizer دارد). الگوریتم با ارزیابی وضعیت‌های انتهایی (برگ‌های درخت) شروع می‌شود و سپس از پایین به بالا امتیازها را برای هر گره محاسبه می‌کند. در هر سطح، بازیکن Maximizer بهترین امتیاز ممکن را انتخاب می‌کند، در حالی که بازیکن Minimizer کمترین امتیاز ممکن را انتخاب می‌کند. این روند تا ریشه درخت ادامه می‌یابد، جایی که حرکت بهینه برای Maximizer انتخاب می‌شود. برای بهبود کارایی، معمولاً از **هرس آلفا – بتا** استفاده می‌شود که شاخه‌هایی از درخت را که در نتیجه تأثیری ندارند، حذف می‌کند.

۱) پیاده سازی الگوریتم Minimax به روش هرس آلفا – بتا:

```python
def minimax(board, depth, alpha, beta, maximizing_player):

    terminal_state = is_terminal_node(board)
    posible_moves = get_valid_locations(board)

    if terminal_state or depth == 0:

        if terminal_state:
            if winning_move(board, AI_PIECE):
                return None, 50
            elif winning_move(board, PLAYER_PIECE):
                return None, -50
            else:
                return None, 0
        else:
            return None, score_position(board, AI_PIECE)

    if maximizing_player:
        best_col = np.random.choice(posible_moves)
        value = -math.inf

        for col in posible_moves:
            row = get_next_open_row(board, col)
            temp_board = board.copy()
            drop_piece(temp_board, row, col, AI_PIECE)
            _, new_score = minimax(temp_board, depth - 1, alpha, beta, maximizing_player: False)
            if new_score > value:
                value = new_score
                best_col = col
            alpha = max(alpha, value)
            if beta <= alpha:
                break
        return best_col, value
```

```
                break
        return best_col, value

    else:  # Minimizing player
        best_col = np.random.choice(posible_moves)
        value = math.inf

        for col in posible_moves:
            row = get_next_open_row(board, col)
            temp_board = board.copy()
            drop_piece(temp_board, row, col, PLAYER_PIECE)
            _, new_score = minimax(temp_board, depth - 1, alpha, beta, maximizing_player: True)
            if new_score < value:
                value = new_score
                best_col = col
            beta = min(beta, value)
            if beta <= alpha:
                break
        return best_col, value
```





این تابع، پیاده‌سازی الگوریتم Minimax با هرس آلفا-بتا است که به دنبال بهترین حرکت برای بازی Connect Four است. ابتدا وضعیت پایانی یا عمق صفر بررسی می‌شود: اگر بازی پایان یافته باشد، امتیاز مثبت (۵۰) برای پیروزی هوش مصنوعی، منفی (۵۰-) برای پیروزی بازیکن، یا صفر برای مساوی بازگردانده می‌شود؛ در غیر این صورت، امتیاز موقعیت فعلی محاسبه می‌شود. سپس، اگر بازیکن Maximizer باشد (هوش مصنوعی)، الگوریتم با شبیه‌سازی تمام حرکات ممکن، بیشترین امتیاز ممکن را محاسبه کرده و آلفا را به‌روزرسانی می‌کند. اگر بازیکن Minimizer باشد (بازیکن انسانی)، کمترین امتیاز ممکن محاسبه شده و بتا به‌روزرسانی می‌شود. برای بهبود کارایی، اگر مقدار آلفا از بتا بیشتر شود (شاخه بی‌اثر)، ادامه پردازش قطع می‌شود. در نهایت، بهترین ستون و امتیاز برای بازیکن بازگردانده می‌شود.

۲) نحوه امتیاز دهی برای بازیکن و هوش مصنوعی در این الگوریتم (قسمت امتیازی):

روش امتیازدهی در این الگوریتم از تابع score_position استفاده می‌کند که موقعیت‌های مختلف برد را ارزیابی می‌کند. برای هوش مصنوعی (AI)، اگر در یک پنجره ۴تایی (row, column یا diagonal) چهار مهره داشته باشد، امتیاز ۱۰۰ کسب می‌کند؛ سه مهره و یک خانه خالی امتیاز ۵ و دو مهره و دو خانه خالی امتیاز ۲ می‌دهد. همچنین اگر بازیکن (حریف) در یک پنجره ۴تایی سه مهره و یک خانه خالی داشته باشد، ۴- امتیاز به عنوان جریمه از AI کسر می‌شود.

برای بازیکن نیز همین تابع استفاده می‌شود، اما مهره‌های بازیکن به عنوان مهره اصلی در نظر گرفته می‌شوند. بنابراین، بازیکن نیز بر اساس تعداد مهره‌های خود در هر پنجره ۴تایی، امتیاز مثبت کسب کرده و حضور مهره‌های AI در این پنجره‌ها به ضرر بازیکن امتیاز منفی محاسبه می‌شود.

نکته: تابع score_position در کد اولیه داده شده وجود داشت و من تنها از آن برا شیوه جدید امتیاز دهی استفاده کردم. همچنین همانطور که در تصاویر خروجی مشخص است با برد هر کدام از طرفین، ۱۰۰۰ امتیاز به او داده می شود.

- **الگوریتم درخت جست و جوی مونت کارلو (MCTS) (امتیازی)**

الگوریتم درخت جست‌وجوی مونت کارلو (**MCTS**) یک الگوریتم مبتنی بر نمونه‌گیری تصادفی است که برای تصمیم‌گیری در محیط‌های نامعین و بازی‌های پیچیده استفاده می‌شود. این الگوریتم شامل چهار مرحله است:

انتخاب (Selection): از ریشه درخت شروع کرده و بر اساس معیارهای اکتشاف و بهره‌برداری (مانند UCB۱)، مسیری از گره‌ها را انتخاب می‌کند تا به گره‌ای برسد که هنوز به‌طور کامل گسترش نیافته است.

گسترش (Expansion): اگر گره انتخابی به‌طور کامل گسترش نیافته نباشد، یک فرزند جدید ایجاد می‌شود که نشان‌دهنده یک حرکت معتبر در بازی است.

شبیه‌سازی (Simulation): از گره جدید، یک بازی تصادفی تا پایان انجام می‌شود (پیشروی تصادفی) و نتیجه بازی (پیروزی، باخت یا تساوی) محاسبه می‌شود.

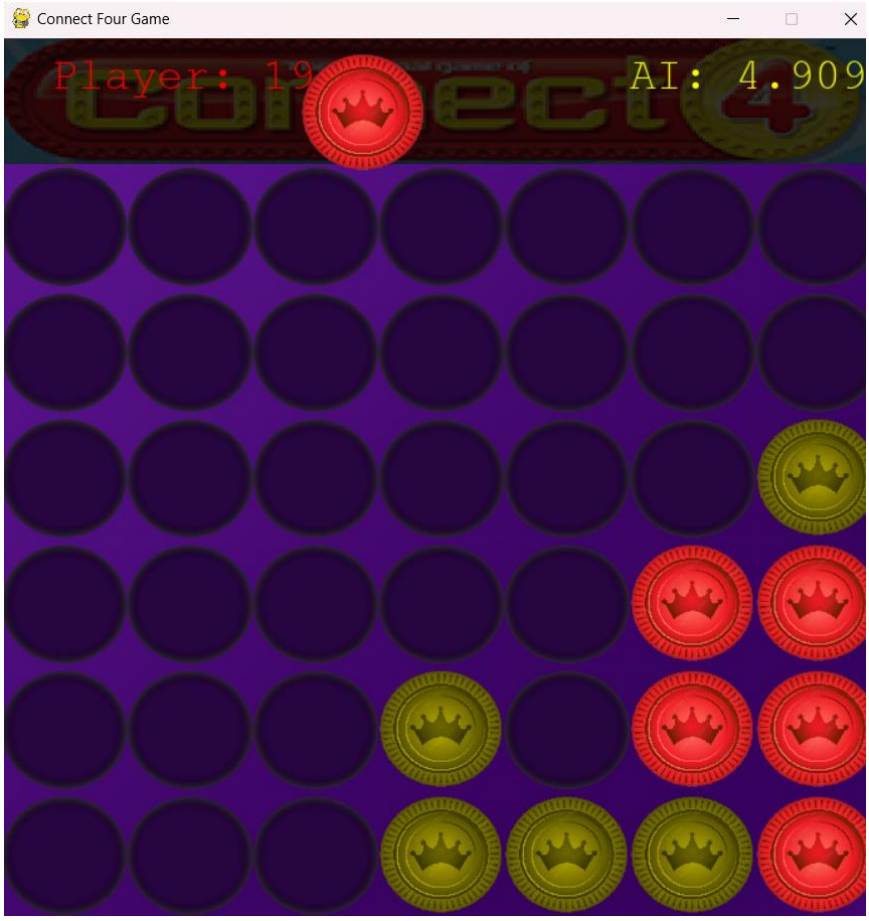
بروزرسانی (Backpropagation): نتیجه شبیه‌سازی به‌صورت بازگشتی به تمام گره‌های مسیر بازگردانده می‌شود و آمار هر گره (تعداد بازدیدها و امتیاز) به‌روزرسانی می‌شود.

این مراحل بارها تکرار می‌شوند تا درخت گسترش یافته و بهترین تصمیم بر اساس ترکیب اکتشاف و بهره‌برداری مشخص شود. گره‌ای که بیشترین نسبت امتیاز به بازدیدها را دارد، به‌عنوان حرکت بهینه انتخاب می‌شود.

۱) پیاده سازی الگوریتم MCTS :

```python
class GameState:
    def __init__(self, board, last_move=None):
        self.board = board
        self.last_move = last_move


    2 usages (2 dynamic)
    def legal_moves(self):

        return [col for col in range(COLUMN_COUNT) if is_valid_location(self.board, col)]


    1 usage
    def tryMove(self, col):

        if is_valid_location(self.board, col):
            return get_next_open_row(self.board, col)
        return None


    1 usage (1 dynamic)
    def terminal(self):

        return is_terminal_node(self.board)


    def winner(self):
```

```python
class Node:
    def __init__(self, state, parent=None):
        self.visits = 1
        self.reward = 0.0
        self.state = state
        self.children = []
        self.children_move = []
        self.parent = parent


    1 usage (1 dynamic)
    def add_child(self, child_state, move):
        child = Node(child_state, self)
        self.children.append(child)
        self.children_move.append(move)


    def update(self, reward):
        self.reward += reward
        self.visits += 1


    def fully_explored(self):
        return len(self.children) == len(self.state.legal_moves())

```


Connect Four Game
Player: 19
AI: 4.909


Game Over
AI Wins!
Player Score: 32
AI Score: 1007.768

برای پیاده‌سازی الگوریتم درخت جست‌وجوی مونت کارلو (MCTS)، دو کلاس اصلی تعریف شده‌اند:

کلاس GameState: وظیفه مدل‌سازی وضعیت بازی را دارد، شامل متدهایی برای تولید حرکات معتبر (legal_moves)، بررسی حرکت نهایی (terminal)، و ایجاد وضعیت جدید پس از حرکت (next_state).

کلاس Node: یک گره درخت را نشان می‌دهد که شامل وضعیت بازی، والد، فرزندان، آمار بازدیدها و امتیاز گره است. این کلاس متدهایی مانند افزودن فرزند (add_child)، بررسی گسترش کامل گره (fully_explored) و به‌روزرسانی آمار گره را فراهم می‌کند.

این کلاس‌ها به‌صورت مجتمع برای ساخت درخت جست‌وجوی مونت کارلو و پیشبرد حرکت‌ها در بازی استفاده می‌شوند.

تابع اصلی الگوریتم MCTS با تعداد تکرارهای مشخص و گره ریشه شروع می‌شود. در هر تکرار، ابتدا گره‌ای با استفاده از سیاست درختی (tree_policy) انتخاب می‌شود. سپس، گره جدیدی از این گره گسترش داده می‌شود (expand). شبیه‌سازی بازی از گره جدید با حرکات تصادفی تا رسیدن به یک وضعیت پایانی انجام می‌شود (default_policy) و نتیجه بازی به‌عنوان امتیاز محاسبه می‌گردد. در نهایت، امتیاز حاصل از شبیه‌سازی در طول مسیر بازگشتی به گره‌های والد برای به‌روزرسانی تعداد بازدیدها و مجموع امتیازات منتقل می‌شود (backup). بهترین گره با توجه به معیار بهره‌برداری و اکتشاف به‌عنوان حرکت انتخاب می‌شود.

۲) نحوه امتیاز دهی برای بازیکن و هوش مصنوعی در این الگوریتم

در الگوریتم MCTS، امتیازدهی بر اساس نتایج شبیه‌سازی بازی انجام می‌شود. در هر شبیه‌سازی (default_policy)، بازی به‌طور تصادفی تا رسیدن به وضعیت پایانی ادامه می‌یابد. در پایان، اگر هوش مصنوعی برنده شود، امتیاز مثبت (معمولاً

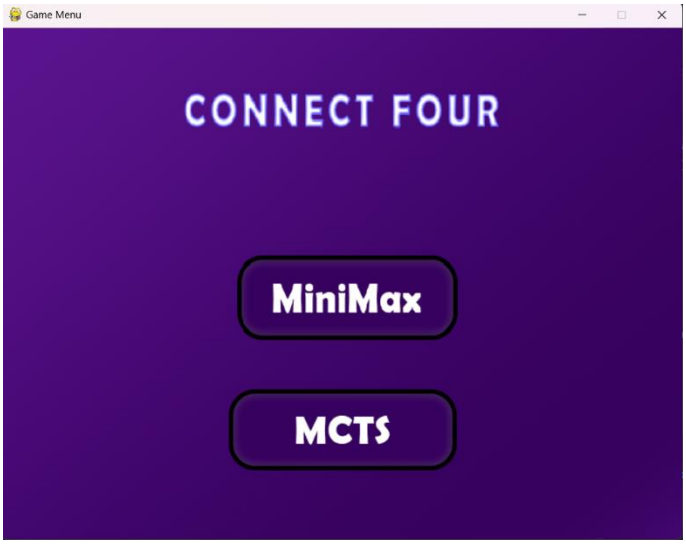
۱) و اگر بازیکن برنده شود، امتیاز منفی (معمولاً ۱-) اختصاص داده می‌شود. این امتیاز در مسیر بازگشتی از گره فرزند به گره والد برای به‌روزرسانی میانگین امتیازات و تعداد بازدیدها استفاده می‌شود.

برای بازیکن، روش امتیازدهی مشابه الگوریتم Minimax است. امتیازات بر اساس ترکیب‌های موجود روی صفحه ارزیابی می‌شود؛ به‌طوری‌که امتیاز بالاتر برای ترکیب‌هایی که منجر به پیروزی بازیکن می‌شوند (مانند چهار مهره در یک ردیف) و امتیاز منفی برای شرایطی که به نفع حریف است، در نظر گرفته می‌شود.

- **مقایسه دو الگوریتم پیاده سازی شده**

در این پروژه، الگوریتم‌های Minimax و MCTS هر دو برای تصمیم‌گیری هوشمندانه در بازی پیاده‌سازی شده‌اند، اما رویکردهای متفاوتی دارند. Minimax با استفاده از جستجوی کامل و ارزیابی دقیق حالت‌ها، تضمین می‌کند که بهترین حرکت را با توجه به عمق مشخص پیدا کند. این روش برای بازی‌هایی با فضای جستجوی محدود (مانند عمق‌های کم) عملکرد بسیار قوی دارد. در مقابل، MCTS مبتنی بر نمونه‌گیری تصادفی و شبیه‌سازی‌های متعدد است که به آن اجازه می‌دهد در بازی‌هایی با فضای جستجوی بزرگ (مانند عمق‌های زیاد یا حالاتی با عدم قطعیت) بهتر عمل کند. از نظر عملکرد، Minimax برای عمق کم و بازی‌های با قوانین مشخص دقیق‌تر است، در حالی که MCTS در بازی‌هایی با پیچیدگی زیاد یا زمانی که فضای جستجو بسیار بزرگ است، انعطاف‌پذیری و کارایی بیشتری دارد.

- **بهبود گرافیک بازی و افزودن منو و انیمیشن**



در پروژه، از تصاویر سفارشی برای پس‌زمینه، مهره‌ها و نوار امتیاز استفاده شده است. این تصاویر با استفاده از کتابخانه Pygame به طور مناسب مقیاس‌بندی شده و در مکان‌های مختلف صفحه نمایش داده می‌شوند، که ظاهر کلی بازی را زیباتر کرده است.

هنگام انداختن مهره در ستون، انیمیشنی طراحی شده که مهره به تدریج از بالای صفحه به موقعیت نهایی خود می‌رسد. این انیمیشن باعث می‌شود حرکات در بازی پویا و جذاب‌تر به نظر برسند و تجربه کاربری بهتری ایجاد شود.

منوی بازی شامل یک صفحه ابتدایی با پس‌زمینه زیبا و تصاویر برای انتخاب الگوریتم هوش مصنوعی (Minimax یا MCTS) است. دکمه‌ها با طراحی گرافیکی مناسب در مرکز صفحه قرار گرفته‌اند و فاصله عمودی کافی بین آن‌ها وجود دارد، که به وضوح و زیبایی منو کمک کرده است.