ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

Bài tập lớn II:

Chess Bot áp dụng giải thuật Monte Carlo Tree Search

Học kì 242 - Lớp L02

Instructor(s): Vương Bá Thịnh

Student(s):

Lý Nguyên Khang	2211437 ***(Leader)***
Phạm Gia Nguyên	2212319
Phan Văn Hội	2211139
Huỳnh Ngọc Duy Khương	2211710

TP. HỒ CHÍ MINH , 04/2025

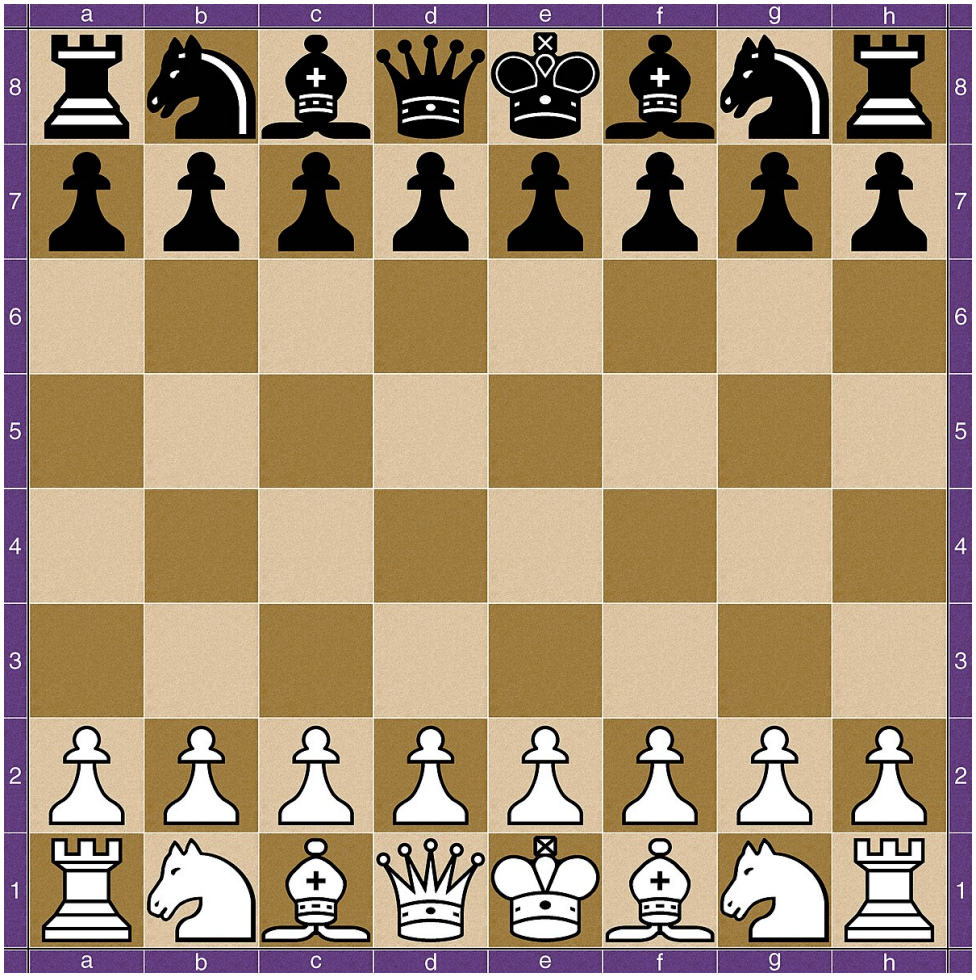
Mục lục

1 Giới thiệu chung

1.1 Giới thiệu đề tài

Cờ vua là một trò chơi chiến thuật lâu đời, được biết đến rộng rãi như một biểu tượng của trí tuệ và sự tư duy logic. Với 64 ô vuông xen kẽ đen trắng và 32 quân cờ chia đều cho hai bên và mục tiêu tối thượng của trò chơi chính là dùng các nước đi tốt nhất để có thể ăn được vua đối phương là thắng.

Cờ vua đòi hỏi người chơi không chỉ có khả năng tính toán trước nhiều nước đi mà còn phải xây dựng chiến lược tổng thể để giành chiến thắng. Trò chơi này có số lượng trạng thái bàn cờ khổng lồ, vượt xa nhiều trò chơi trí tuệ khác, khiến cho việc tìm kiếm nước đi tối ưu trở thành một thách thức lớn đối với cả con người và máy tính.



Hình 1: Bàn cờ vua

Xuất phát từ sự hấp dẫn và độ phức tạp của cờ vua, đề tài của chúng tôi hướng tới việc xây dựng một con bot chơi cờ vua sử dụng giải thuật tìm kiếm Monte Carlo Tree Search (MCTS). Đây là một thuật toán nổi tiếng trong lĩnh vực trí tuệ nhân tạo, đặc biệt hiệu quả trong các bài toán có không gian trạng thái lớn và khó xác định chiến lược tối ưu ngay lập tức. Bằng cách mô phỏng nhiều ván chơi ngẫu nhiên từ các trạng thái hiện tại và thống kê kết quả, MCTS giúp bot lựa chọn nước đi có xác suất chiến thắng cao

nhất. Đề tài không chỉ nhằm tạo ra một đối thủ ảo có khả năng chơi cờ mạnh mẽ mà còn là cơ hội để nhóm nghiên cứu sâu hơn về ứng dụng thực tiễn của các kỹ thuật tìm kiếm hiện đại trong các trò chơi trí tuệ.

1.2 Luật cơ bản của cờ vua

- **Bàn cờ:**
  - 8 hàng (1–8) và 8 cột (a–h), xen kẽ ô đen trắng.
  - Mỗi bên có 16 quân: 1 vua, 1 hậu, 2 xe, 2 tượng, 2 mã và 8 tốt.
- **Mục tiêu:**
  - Chiếu hết (*checkmate*) vua đối phương.
- **Cách di chuyển các quân cờ:**
  - **Vua:** Đi 1 ô theo mọi hướng.
  - **Hậu:** Đi ngang, dọc hoặc chéo không giới hạn ô.
  - **Xe:** Đi ngang hoặc dọc không giới hạn ô.
  - **Tượng:** Đi chéo không giới hạn ô.
  - **Mã:** Đi theo hình chữ "L" (2 ô theo một hướng, rồi 1 ô vuông góc).
  - **Tốt:**
    - Đi thẳng 1 ô (2 ô ở nước đầu tiên).
    - Ăn chéo 1 ô phía trước.
- **Các quy tắc đặc biệt:**
  - **Nhập thành (Castling):**
    - Vua và một xe cùng di chuyển trong một nước đi: vua đi 2 ô về phía xe, xe nhảy qua đứng kế vua.
    - Điều kiện:
      - Vua và xe chưa từng di chuyển.
      - Không có quân cản giữa vua và xe.
      - Vua không đang bị chiếu, không đi qua ô bị chiếu.
  - **Ăn tốt qua đường (En passant):**
    - Khi một tốt đi 2 ô từ vị trí ban đầu và đứng cạnh tốt đối phương, đối phương có quyền ăn chéo như thể tốt đó chỉ đi 1 ô.
    - Phải thực hiện ngay trong nước đi kế tiếp.
  - **Phong cấp (Promotion):**
    - Khi tốt đi đến hàng cuối cùng, nó được phong thành hậu, xe, tượng hoặc mã (thường chọn hậu).
- **Kết thúc ván cờ:**

– Chiếu hết vua đối phương.
– Bị chiếu và không còn nước đi hợp lệ.
– Hòa: đồng ý hòa, hết nước đi, hoặc lặp lại trạng thái.

2 Cơ sở lý thuyết

2.1 Monte Carlo Tree Search

Phương pháp Monte Carlo là một kỹ thuật tính toán dựa trên việc sử dụng các phép thử ngẫu nhiên để giải quyết các bài toán phức tạp mà khó có thể giải trực tiếp bằng các phương pháp phân tích truyền thống. Được đặt tên theo thành phố Monte Carlo nổi tiếng với các sòng bạc, phương pháp này tận dụng tính ngẫu nhiên để mô phỏng và ước lượng các kết quả tiềm năng trong một không gian trạng thái lớn.

Trong lĩnh vực trí tuệ nhân tạo và trò chơi, phương pháp Monte Carlo được ứng dụng chủ yếu thông qua thuật toán Monte Carlo Tree Search (MCTS). MCTS là một giải thuật tìm kiếm quyết định, kết hợp giữa việc xây dựng cây trạng thái và sử dụng mô phỏng ngẫu nhiên để đánh giá chất lượng của các hành động. Thay vì đánh giá toàn bộ không gian trạng thái (vốn rất lớn trong những trò chơi như cờ vua, cờ vây), MCTS chỉ tập trung vào những nhánh khả thi nhất, từ đó dần dần tối ưu quyết định.

Một chu kỳ cơ bản của Monte Carlo Tree Search bao gồm bốn bước:

1. **Selection (Chọn lựa):** Từ gốc, lựa chọn liên tiếp các hành động theo một chiến lược cân bằng giữa khám phá và khai thác, thường dùng công thức UCT (Upper Confidence Bound for Trees):

$$\mathrm{UCT}_i = \frac{w_i}{n_i} + c \cdot \sqrt{\frac{\ln N}{n_i}}$$

   Trong đó:

   - w_i: tổng phần thưởng của nút con i
   - n_i: số lần nút con i được chọn
   - N: tổng số lượt viếng thăm của nút cha
   - c: hệ số điều chỉnh giữa khai thác và khám phá (thường chọn $c = \sqrt{2}$)

2. **Expansion (Mở rộng):** Nếu nút hiện tại không phải là nút kết thúc trò chơi và còn có hành động chưa được mở rộng, thêm một hoặc vài nút con mới vào cây tìm kiếm.

3. **Simulation (Mô phỏng):** Từ nút mới mở rộng, thực hiện mô phỏng ngẫu nhiên trò chơi đến khi kết thúc để thu thập kết quả (thắng, thua hoặc hòa). Kết quả mô phỏng được biểu diễn dưới dạng phần thưởng $r \in \mathbb{R}$.

4. **Backpropagation (Lan truyền ngược):** Cập nhật các nút trên đường đi từ nút mới về gốc dựa trên kết quả mô phỏng:

$$n_i \leftarrow n_i + 1, \quad w_i \leftarrow w_i + r$$

Sau khi thực hiện nhiều vòng lặp MCTS, hành động được chọn tại nút gốc là hành động có số lượt viếng thăm cao nhất:

$$a^* = \arg\max_{i \in \text{children}} n_i$$

Qua nhiều lượt lặp lại, cây tìm kiếm sẽ tập trung phát triển ở những vùng có khả năng thắng cao nhất, giúp bot đưa ra nước đi tối ưu. MCTS đặc biệt phù hợp với các trò chơi có không gian trạng thái khổng lồ và độ không chắc chắn cao, bởi vì nó không cần đánh giá toàn bộ trạng thái một cách chính xác.

2.2 Mô phỏng trạng thái

Chúng tôi sẽ thiết lập bàn cờ vua, với các quân trên bàn ở một vị trí xác định, mỗi cấu hình như vậy được biểu diễn dưới dạng một **trạng thái** và tương ứng với một **nút** trong cây tìm kiếm. Mỗi trạng thái mang đầy đủ thông tin về vị trí của tất cả các quân, lượt đi hiện tại (trắng hoặc đen), và các điều kiện đặc biệt như quyền nhập thành, bắt tốt qua đường (en passant), hay tình trạng chiếu.

Từ một nút (trạng thái hiện tại), có thể sinh ra các nút con tương ứng với tất cả những **nước đi hợp lệ** tiếp theo theo luật chơi cờ vua. Các nước đi hợp lệ này được xác định dựa trên quy tắc di chuyển riêng biệt của từng loại quân như sau:

- **Tốt (Pawn):** di chuyển một ô phía trước nếu ô đó trống, hoặc hai ô nếu ở hàng xuất phát; bắt quân theo đường chéo. Có thể thực hiện bắt tốt qua đường (en passant) và phong cấp khi đến hàng cuối.
- **Xe (Rook):** di chuyển theo hàng ngang hoặc dọc không giới hạn số ô, miễn là không bị chắn.
- **Mã (Knight):** di chuyển theo hình chữ "L", tức là hai ô theo một hướng và một ô theo hướng vuông góc; có thể nhảy qua quân khác.
- **Tượng (Bishop):** di chuyển theo đường chéo không giới hạn số ô, miễn là không bị chắn.
- **Hậu (Queen):** kết hợp khả năng của Xe và Tượng: di chuyển theo hàng, cột hoặc chéo.
- **Vua (King):** di chuyển một ô theo mọi hướng; có thể nhập thành nếu không bị chiếu và các điều kiện nhập thành được thoả mãn.

Tuy nhiên các nước đi hợp lệ với những điều trên đồng thời cũng phải đảm bảo quân vua 2 bên được an toàn trong xuyên suốt quá trình đi. Việc di chuyển dẫn đến vua bị ăn ngay là một nước đi không hợp lệ và không thể sinh ra tiếp các nước tiếp theo.

Quá trình mô phỏng trạng thái là việc tạo ra các trạng thái con từ trạng thái hiện tại, tương ứng với từng nước đi hợp lệ. Trong Monte Carlo Tree Search, một mô phỏng có thể thực hiện nhiều lượt đi ngẫu nhiên liên tiếp, từ trạng thái hiện tại đến khi ván cờ kết thúc (chiếu hết, hòa, hoặc hết nước đi), nhằm đánh giá chất lượng tiềm năng của trạng thái ban đầu.

Tuy nhiên tập dữ liệu khá lớn và rất lâu để sinh và tự thực thi lại vì vậy chúng tôi sử dụng thư viện python-chess. Đây là một thư viện chuyên về chess, giúp chúng tôi dễ dàng khởi tạo bàn cờ , thực hiện các nước đi cũng như lịch sử đi , đồng thời cũng kiểm tra giùm các nước đi hợp lệ và các nước nhập thành, chiếu tướng và phong tốt,... Nhờ thư viện chúng tôi sẽ tập trung vào thiết kế thuật toán MCTS mà không cần để ý đến các logic của cờ vua quá nhiều

3 Mô hình toán học - Thực thi

3.1 Mô hình toán học

Trong bài toán thiết kế agent chơi cờ vua sử dụng Monte Carlo Tree Search (MCTS), chúng tôi xây dựng một mô hình toán học bao gồm các thành phần sau:

- **Không gian trạng thái (State Space)**: Tập tất cả các cấu hình hợp lệ của bàn cờ. Mỗi trạng thái được biểu diễn bằng đối tượng `chess.Board` từ thư viện `python-chess`.
- **Không gian hành động (Action Space)**: Tập hợp các nước đi hợp lệ từ mỗi trạng thái, biểu diễn bằng đối tượng `chess.Move`.
- **Hàm chuyển trạng thái (Transition Function)**: Cho trạng thái hiện tại s và hành động a, trạng thái mới $s' = T(s, a)$ được tính bằng cách gọi `board.push(move)`.
- **Hàm đánh giá đầu cuối (Rollout/Simulation)**: Từ nút lá, thực hiện các nước đi ngẫu nhiên đến khi kết thúc ván cờ, sau đó đánh giá kết quả thắng/thua/hòa.
- **Hàm lựa chọn và mở rộng (Selection & Expansion)**: Dựa trên công thức UCT:

$$\text{UCT}(i) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

  Trong đó:

  - w_i: số lần chiến thắng tại nút i
  - n_i: số lần nút i được thăm
  - N: tổng số lượt thăm nút t
  - c: tham số cân bằng giữa khai thác (exploitation) và thăm dò (exploration)

3.2 Thư viện python-chess

python-chess là một thư viện Python mã nguồn mở hỗ trợ đầy đủ luật chơi cờ vua, bao gồm sinh nước đi hợp lệ, kiểm tra chiếu tướng, phong hậu, nhập thành và lưu trữ trạng thái dưới dạng PGN hoặc FEN. Nhờ sử dụng thư viện này, chúng tôi có thể tập trung vào thiết kế thuật toán MCTS mà không cần xử lý logic cờ vua phức tạp.

3.3 Khởi tạo node trong cây tìm kiếm MCTS

Dưới đây là đoạn mã khởi tạo của lớp MCTSNode, đại diện cho một node trong cây tìm kiếm:

```
class MCTSNode:
    def __init__(self, board, difficulty, parent=None, move=None):
        self.board = board
        self.parent = parent
        self.move = move
        self.children = []
        self.visits = 0
        self.wins = 0
        self.untried_moves = list(board.legal_moves)
        self.prioritize_moves()
        self.difficulty = difficulty
```

- board: trạng thái bàn cờ tại node hiện tại, được sao chép từ đối tượng chess.Board
- parent: node cha trong cây MCTS
- move: nước đi được thực hiện để đến node này
- children: danh sách các node con được mở rộng từ node hiện tại
- visits và wins: số lượt thăm và số lần chiến thắng tại node này, phục vụ tính toán UCT
- untried_moves: danh sách các nước đi chưa được thử từ trạng thái hiện tại
- prioritize_moves(): hàm ưu tiên các nước đi quan trọng như chiếu, ăn quân hoặc kiểm soát trung tâm, hỗ trợ cải thiện hiệu quả MCTS theo cấp độ khó
- difficulty: các thông số độ khó, ảnh hưởng đến cách agent ra quyết định

3.4 Mô phỏng công thức

3.4.1 Hàm simulation

Hàm **evaluate_position** nhằm đánh giá một trạng thái cờ vua dựa trên các yếu tố vật chất, vị trí, và các đe dọa, từ đó đưa ra một giá trị đánh giá cho trạng thái này. Đây cũng là hàm mà chúng tôi sẽ đánh giá đầu cuối hay **Simulation** từ nút trạng thái lá. Vì

việc không gian duyệt quá lớn nên chúng tôi quyết định đánh giá nó bằng số điểm trạng thái bàn cờ ngay lúc đó, xem lợi thế đang nghiêng về bên trắng hay bên đen , từ đó giúp đánh giá chính xác hơn mực độ lợi thế của bàn cờ và điều đó dễ giúp tìm ra đường thắng hơn.

```python
@lru_cache(maxsize=10000)
def evaluate_position(board_fen):
    """Evaluate a chess position (higher values favor white)"""
    board = chess.Board(board_fen)

    if board.is_checkmate():
        return -1000 if board.turn else 1000
    if board.is_stalemate() or board.is_insufficient_material():
        return 0

    # Material counting
    material_score = 0
    pieces = board.piece_map()

    for square, piece in pieces.items():
        value = get_piece_value(piece)
        if piece.piece_type == chess.PAWN:
            rank = chess.square_rank(square)
            bonus = rank - 1 if piece.color == chess.WHITE else 6 - rank
            value += bonus * 0.1

        factor = 1 if piece.color == chess.WHITE else -1
        material_score += factor * value

    if board.is_check():
        if board.turn == chess.WHITE:
            material_score -= 1  # White is in check
        else:
            material_score += 1  # Black is in check

    mobility = len(list(board.legal_moves))
    board_copy = board.copy()
    board_copy.push(chess.Move.null())
    opponent_mobility = len(list(board_copy.legal_moves))
    mobility_score = 0.05 * (mobility - opponent_mobility)

    if board.turn == chess.WHITE:
        material_score += mobility_score
    else:
        material_score -= mobility_score

    return material_score
```

Listing 1: Hàm evaluate_position đánh giá vị trí trên bàn cờ

Giải thích chi tiết

Cụ thể, hàm này thực hiện các bước sau:

1. **Kiểm tra kết thúc ván cờ:**
   - Nếu ván cờ kết thúc với chiếu tướng (checkmate), giá trị trả về là `-1000` nếu là lượt của Trắng và `1000` nếu là lượt của Đen.
   - Nếu ván cờ kết thúc với hòa (stalemate) hoặc không có đủ quân để chiến thắng (insufficient material), giá trị trả về là `0`.
2. **Đánh giá vật chất (Material Counting):**
   - Đo lường sức mạnh của các quân cờ bằng cách sử dụng các giá trị cụ thể cho từng quân cờ. Ví dụ, quân Xe có giá trị lớn hơn quân Tốt.
   - Đo lường giá trị vật chất của các quân cờ như sau:

$$\text{piece\_values} = \left\{ \begin{array}{lll} \text{PAWN} & : 2, & \text{tốt} \\ \text{KNIGHT} & : 6, & \text{mã} \\ \text{BISHOP} & : 6, & \text{tượng} \\ \text{ROOK} & : 10, & \text{xe} \\ \text{QUEEN} & : 18, & \text{hậu} \\ \text{KING} & : 0 & \text{vua} \end{array} \right\}$$

   - Dựa trên bảng quân cờ hiện có, tính toán tổng giá trị vật chất của cả hai bên.
   - Cải thiện đánh giá cho các quân Tốt khi tiến vào gần hàng cuối của đối phương (tạo ra bonus cho các quân Tốt gần với hàng cuối - Phong cấp).
3. **Kiểm tra các mối đe dọa (Threats):**
   - Nếu một bên đang bị chiếu (check), giảm hoặc tăng điểm đánh giá của bên đó tùy thuộc vào lượt đi.
4. **Đánh giá khả năng di chuyển (Mobility):**
   - Đánh giá khả năng di chuyển của quân cờ, tức là số lượng các nước đi hợp lệ có sẵn.
   - So sánh số nước đi hợp lệ của Trắng và Đen, và sử dụng sự khác biệt này để điều chỉnh điểm đánh giá.
5. **Tổng hợp điểm số (Score Aggregation):**
   - Tổng hợp tất cả các điểm số đã tính được qua các tiêu chí trên và trả về một giá trị tổng thể cho vị trí hiện tại trên bàn cờ.

3.4.2 Thiết lập vòng lặp

Ở đây chúng tôi sẽ thiết lặp vòng lặp cho giải thuật Monte Carlo Tree Search. Mục tiêu của vòng lặp này là duyệt qua một số lượng lớn các lượt mô phỏng (simulations) để chọn ra nước đi tốt nhất có thể trong tình huống hiện tại của bàn cờ. **Cấu trúc vòng lặp chính**

```
    while iterations < len(list(board.legal_moves))**iter_limit:
```

Vòng lặp này được điều khiển bởi giá trị iterations, tức là số lần thuật toán thực hiện mô phỏng MCTS.

Điểm đặc biệt: Số vòng lặp không cố định mà phụ thuộc vào số lượng nước đi hợp lệ hiện tại (len(list(board.legal_moves))) và giá trị iter_limit truyền vào. Điều này giúp chúng tôi đảm bảo là sẽ giúp cho mô hình sẽ duyệt được qua các node có thể di chuyển bằng cách lấy số nước đi có thể và lũy thừa với trọng số truyền vào.

Cách tính len(list(board.legal_moves))**iter_limit cho phép số vòng lặp tăng nhanh theo cả độ phức tạp của bàn cờ và độ khó được thiết lập. Độ khó sẽ quyết định đến số lần duyệt của cây, ví dụ như độ khó dễ, cây chỉ quyết số vòng lặp khá nhỏ, còn đối với độ khó trung bình cây sẽ duyệt nhiều hơn , giúp bot có thể dự đoán trước được 1,2 nước của đối thủ để phòng chống và ngăn chặn đối thủ của mình.

Ví dụ cụ thể như sau:

Giả sử có 20 nước đi hợp lệ và đặt `iter_limit` = 2, thì số vòng lặp tối đa là:

$$20^2 = 400$$

Nếu `iter_limit` = 3, thì số vòng lặp tối đa sẽ là:

$$20^3 = 8000$$

Điều này cho phép điều chỉnh mức độ thông minh của thuật toán MCTS tùy theo độ khó của trò chơi. Trò chơi càng khó thì `iter_limit` càng lớn, và số vòng lặp càng nhiều, từ đó cải thiện chất lượng lựa chọn nước đi. Ngược lại, với độ khó thấp, số vòng lặp sẽ giảm, giúp tiết kiệm tài nguyên tính toán.

3.5 Vòng lặp chính

Ở đây thuật toán để tạo ra con bot này sẽ xoay quanh vòng lặp chính của giải thuật Monte Carlo Tree Search. Như đã giải thích ở trên, số vòng lặp để duyệt qua các node sẽ tuân thủ theo độ khó mà bạn chọn, từ đó số vòng lặp sẽ tùy thuộc vào giá trị bạn truyền vào và số lượng nước đi hợp lệ lúc đó. Từ đó sẽ ra quyết định chuẩn xác hơn.

```
    def mcts(board, iter_limit=1000, time_limit=20.0, c_param=1.4):
    root = MCTSNode(board, c_param)
    start_time = time.time()
    iterations = 0
```

```
    # Use either iteration limit or time limit
    while iterations < len(list(board.legal_moves))**iter_limit:
        # Selection
        node = root
        while node.is_fully_expanded() and node.children:
            node = node.best_child()
            if node is None:  # Safety check
                break

        # Expansion
        if node and not node.is_fully_expanded() and not node.
board.is_game_over():
            node = node.expand()

        # Simulation
        if node:
            result = simulate_smart_game(node.board)
            temp_node_board = node.board.copy()
            # Backpropagation
            while node:
                node.update(result if node.board.turn ==
temp_node_board.turn else  -result)
                node = node.parent

        iterations += 1

    # Choose the best move
    if not root.children:
        # If no children (shouldn't happen in normal play), pick a
    random move
        return random.choice(list(board.legal_moves))

    # Return move with best win rate for serious play
    best_child = max(root.children, key=lambda n: n.wins/n.visits
+ root.difficulty * math.sqrt(math.log(root.visits) / n.visits)
     if n.visits > 0 else 0)
    return best_child.move
```

Chi tiết vào việc thực thi nó sẽ tuân thủ 4 bước giống như giải thuật Monte Carlo Tree Search đã đề cập, đầu tiên chúng ta sẽ chọn ra node con tốt nhất từ trạng thái đầu, node con này được chọn theo công thức:

$$\mathrm{UCT}(i) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

Trong đó:

- w_i: số điểm thắng tích lũy tại nút i
- n_i: số lần nút i được thăm

- N: tổng số lượt thăm nút cha
- c: tham số cân bằng giữa khai thác (exploitation) và thăm dò (exploration)

Chi tiết hơn thì w_i sẽ là tích lũy điểm lợi thế của node đó , và số điểm lợi thế cũng được đề cập và nói đến ở trên nhằm có thể đánh giá sơ bộ thế cục, tránh quét hết cả bàn cờ.

Sau khi chọn node con tốt nhất theo công thức **UCT**, thuật toán sẽ tiếp tục qua ba bước còn lại:

- **Expansion (Mở rộng):** Sau khi chọn được một node không phải lá (hoặc node chưa được mở rộng đầy đủ), nếu vẫn còn nước đi hợp lệ chưa được xét tại node đó, thuật toán sẽ thực hiện mở rộng bằng cách chọn một nước đi chưa thử để tạo thành node con mới. Điều này giúp cây tìm kiếm dần dần bao phủ thêm các trạng thái mới, từ đó mở rộng kiến thức về thế cờ hiện tại.
- **Simulation (Mô phỏng):** Từ node mới được tạo ra, thuật toán sẽ thực hiện mô phỏng một ván chơi từ vị trí hiện tại cho đến khi ván đấu kết thúc hoặc đạt đến một trạng thái dừng nhất định. Trong trường hợp này, hàm `simulate_smart_game()` được sử dụng để mô phỏng theo cách hợp lý hơn thay vì chọn ngẫu nhiên hoàn toàn. Điều này giúp đánh giá chính xác hơn thế cờ mà không cần phải chơi hết toàn bộ ván đấu.
- **Backpropagation (Lan truyền ngược kết quả):** Kết quả từ quá trình mô phỏng sẽ được lan truyền ngược trở lại từ node con vừa được mô phỏng về đến gốc của cây. Tại mỗi node trên đường đi này, số lượt thăm và điểm thắng sẽ được cập nhật. Việc cập nhật điểm thắng được thực hiện theo hướng phù hợp với lượt chơi tại mỗi node:

$$\text{result}_{\text{node}} = \begin{cases} +\text{result} & \text{nếu lượt chơi của node trùng với lượt chơi trong mô phỏng} \\ -\text{result} & \text{ngược lại} \end{cases}$$

Sau khi thực hiện một số vòng lặp nhất định (theo giới hạn độ khó hoặc thời gian), thuật toán sẽ chọn node con tốt nhất từ gốc theo công thức được điều chỉnh như sau:

$$\text{Score}(i) = \frac{w_i}{n_i} + d \cdot \sqrt{\frac{\ln N}{n_i}}$$

trong đó d là tham số điều chỉnh theo độ khó người dùng chọn, cho phép tăng mức độ khám phá hoặc thiên về chiến lược an toàn.

- Nếu không có node con nào được mở rộng (trường hợp bất thường), thuật toán sẽ chọn ngẫu nhiên một nước đi hợp lệ từ bàn cờ hiện tại.
- Nếu có các node con, thuật toán sẽ chọn node có tỉ lệ thắng cao nhất kèm theo phần khám phá, đảm bảo rằng hành vi của AI vừa có tính chắc chắn vừa có sự đổi mới khi cần.

Tóm lại, toàn bộ quá trình này bao gồm bốn bước: **Selection, Expansion, Simulation, Backpropagation** và sẽ được lặp lại cho đến khi đạt đủ số vòng theo độ khó. Đây là nền tảng chính giúp bot ra quyết định hợp lý khi chơi cờ.

4 Kiểm tra và phân cấp độ khó

4.1 Điều chỉnh độ khó cho Agent

Để điều chỉnh độ khó cho agent trong trò chơi, chúng tôi thiết kế một cơ chế cho phép thuật toán MCTS thay đổi hành vi tìm kiếm thông qua hai tham số chính:

- **Số vòng lặp trung bình (difficulty_iterations):** Tham số này xác định số lượt lặp (iterations) mà thuật toán MCTS sẽ thực hiện. Cụ thể, số vòng lặp được tính theo công thức:
$$\text{iterations} = (|\mathcal{M}|)^{\text{difficulty\_iterations[level]}}$$
trong đó $|\mathcal{M}|$ là số lượng nước đi hợp lệ hiện tại (tức là số node con từ trạng thái hiện tại). Công thức này cho phép số vòng lặp tăng theo cấp số mũ dựa trên độ khó và độ phức tạp của trạng thái trò chơi.

- **Tham số UCT (difficulty_cparams):** Đây là hệ số c trong công thức UCT, giúp điều chỉnh sự cân bằng giữa khai thác (*exploitation*) và khám phá (*exploration*). Khi độ khó tăng, giá trị c tăng theo, cho phép agent khám phá nhiều nước đi tiềm năng hơn thay vì chỉ tập trung vào những nước đi có xác suất thắng cao hiện tại.

```
self.difficulty_iterations = {
    "Easy": 1.5,
    "Medium": 1.8,
    "Hard": 2.1
}
self.difficulty_cparams = {
    "Easy": 1.44,
    "Medium": 2.0,
    "Hard": 2.44
}
```

Listing 2: Thiết lập tham số độ khó

Giải thích thêm:

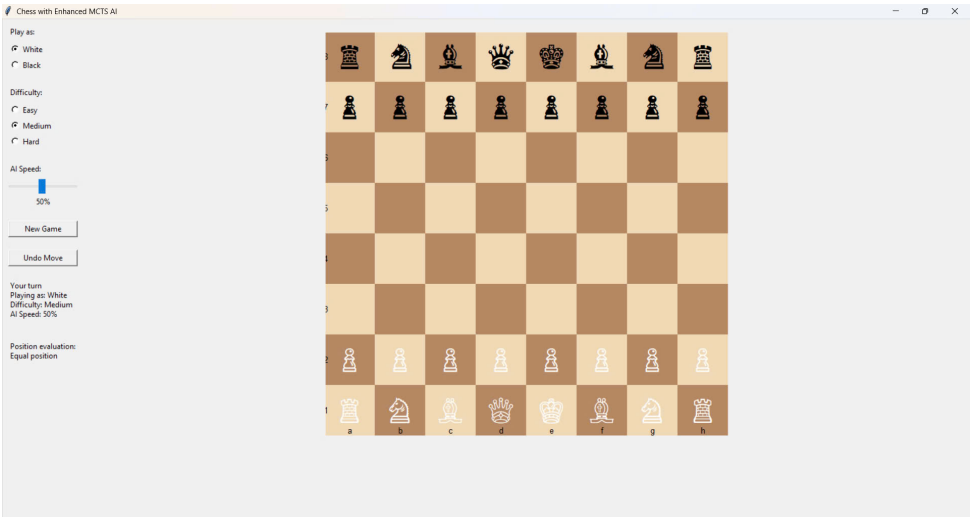
- Ở mức **Easy**, số vòng lặp là tương đối thấp do hệ số lũy thừa nhỏ. Đồng thời, tham số c cũng thấp, dẫn đến agent có xu hướng chọn những nước đi đã được chứng minh hiệu quả, tạo ra các chiến thuật đơn giản, dễ đoán.

- Ở mức **Medium**, cả số vòng lặp và mức độ khám phá được nâng cao, giúp agent có thể cân bằng giữa chiến lược ngắn hạn và dài hạn.

- Ở mức **Hard**, số vòng lặp tăng mạnh theo cấp số mũ với số nước đi hợp lệ. Cùng với hệ số c cao, agent có thể thực hiện tìm kiếm sâu hơn và đánh giá chiến lược dài hạn tốt hơn, từ đó đưa ra các nước đi phức tạp, khó đoán và hiệu quả hơn trong thế cờ.

Nhờ sự điều chỉnh linh hoạt của hai tham số này, hệ thống chatbot chơi cờ có thể cung cấp trải nghiệm phù hợp cho người chơi ở nhiều trình độ khác nhau, từ người mới bắt đầu đến người chơi nâng cao.

5 Thiết lập UI



Hình 2: Giao diện người dùng của ứng dụng chơi cờ vua sử dụng AI

Giao diện ứng dụng được thiết kế trực quan, tập trung vào trải nghiệm người chơi:

- **Chính giữa**: màn hình bàn cờ vua tương tác, nơi người chơi và agent thực hiện nước đi.
- **Góc bên trái**: có các lựa chọn để điều chỉnh độ khó của agent, chọn màu quân cờ mà người chơi điều khiển (trắng hoặc đen), màu còn lại sẽ thuộc về AI.
- **Tùy chọn điều khiển**: cho phép tạo một ván đấu mới, hoặc quay lại các nước đi trước đó.
- **Thông báo trạng thái**: hiển thị phía dưới cùng, cung cấp thông tin về lượt đi hiện tại (phe nào đang đánh) và một số thông tin trạng thái khác của ván cờ.

6 Tài liệu tham khảo