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)

Báo cáo Bài tập lớn 2

GAME PLAYING

Giảng viên hướng dẫn: Vương Bá Thịnh
Nhóm:
Sinh viên: Nguyễn Phạm Quốc An - 2210020
Nguyễn Thùy Dung - 2210487
Nguyễn Trường Thịnh - 2213298
Lê Trọng Thiện - 2313233

Thành phố Hồ Chí Minh, Ngày 4 tháng 5 năm 2025

Mục lục

1 Giới thiệu và mô tả về bài toán

1.1 Giới thiệu trò chơi cờ vua

Cờ vua là trò chơi 2 người, thể loại *zero-sum*. Một người chơi sẽ là bên quân cờ màu trắng và người còn lại sẽ là bên quân cờ đen. Trò chơi sử dụng một bàn cờ hình vuông chia thành 64 ô vuông nhỏ hơn với 8 hàng ngang và 8 hàng dọc. Mỗi người chơi sẽ bắt đầu với 16 quân cờ, bao gồm 8 con tốt, 2 mã, 2 tượng, 2 xe, 1 hậu và 1 vua.

Mặc dù chỉ có 64 ô và 32 quân cờ trên bàn cờ nhưng số lượng nước đi có thể được thì còn vượt xa cả số lượng các nguyên tử có trong vũ trụ.

1.2 Bài toán cần giải quyết

Xây dựng game cờ vua có sử dụng thuật toán trí tuệ nhân tạo, bao gồm:

- Biểu diễn trạng thái bàn cờ sao cho đầy đủ thông tin về vị trí từng quân, lượt đi, quyền đặc biệt (castling, en passant) và trạng thái kiểm chiếu.
- Sinh tập hợp nước đi hợp lệ từ mỗi trạng thái: bao gồm nước đi tiêu chuẩn, bắt quân, thăng cấp quân tốt, en passant và castling, đồng thời phải loại trừ các nước đi khiến chính vua của mình bị chiếu.
- Tính toán nước đi tốt nhất bằng thuật toán Minimax kết hợp alpha-beta pruning trên cây tìm kiếm.

1.3 Mục tiêu, kết quả mong muốn

- Hiện thực game playing agent cho một trò chơi tự chọn (dạng đối kháng).
- Thắng được agent chơi ngẫu nhiên đúng luật (10/10), nghĩa là đấu 10 ván bất kể đi trước hay đi sau đều phải thắng.
- Agent có cấp độ chơi (dễ - trung bình - khó).

2 Các phương pháp áp dụng để giải quyết bài toán

2.1 Thuật toán Minimax

Giải thuật Minimax là một kỹ thuật ra quyết định trong trí tuệ nhân tạo, đặc biệt hữu ích trong các trò chơi hai người như cờ vua, cờ caro, hoặc cờ tướng. Mục tiêu của thuật toán là tối đa hóa lợi ích cho người chơi hiện tại (Maximizer) và tối thiểu hóa lợi ích của đối thủ (Minimizer), giả định rằng cả hai đều chơi tối ưu.

2.1.1 Nguyên lý chung

Thuật toán Minimax được dùng trong trò chơi hai người, một bên muốn tối đa điểm, bên kia muốn tối thiểu điểm.
Ta cần xây dựng một cây trạng thái để biểu diễn bài toán với mỗi node là một trạng thái của bàn cờ và mỗi cạnh là một nước đi.
Ta có hai loại node là Min-node và Max-node:
+ Max-node (đến lượt AI): chọn nước đi có giá trị lớn nhất.
+ Min-node (đến lượt đối thủ): chọn nước đi có giá trị nhỏ nhất.
Ta sẽ đệ quy xuống tới độ sâu xác định hoặc tới trạng thái tận cùng, đánh giá (heuristic) giá trị tại node lá, rồi truyền giá trị đó ngược lên.

2.1.2 Alpha-Beta Pruning

Ta sẽ dùng hai giá trị alpha (giá trị tốt nhất Max đã có) và beta (giá trị tốt nhất Min đã có) để thực hiện cắt tỉa cây trạng thái.
Khi tại một Max-node, nếu alpha $\geq$ beta, tức là nhánh còn lại không thể làm tốt hơn nên ta sẽ tiến hành cắt tỉa nhánh đó của cây hành vi.
Tương tự tại Min-node: nếu beta $\leq$ alpha thì ta cũng sẽ tiến hành cắt tỉa những nhánh không cần thiết đó.

2.2 Các hàm heuristic áp dụng

2.2.1 Tổng điểm từng quân cờ

Trong chương trình cờ vua hiện tại, sử dụng bảng điểm sau:

Quân cờ	**Giá trị**
Tốt	100
Mã	320
Tượng	330
Xe	500
Hậu	900
Vua	**20000**

Bảng 1: Source: Simplified Evaluation Function – Chessprogramming wiki

Điểm tối đa của một bên dựa vào tổng điểm từng quân cờ là:

$$20000 + 9 \times 900 + 2 \times 500 + 4 \times 300 = 30300.$$

2.2.2 Điểm theo vị trí của từng quân

Tốt (Pawn)

Bảng điểm của quân Tốt ưu tiên việc tiến lên, đặc biệt là trên các cột trung tâm, nhằm khuyến khích áp lực lên vị trí đối phương. Những quân Tốt ở hàng hai được thưởng điểm cao để duy trì sự vững chắc trước Vua sau khi nhảy thành. Các ô ở hàng bốn năm (`d4`, `e4`, `d5`, `e5`) có giá trị tích cực hơn so với hàng thấp hoặc hàng rất cao, phản ánh tầm quan trọng của kiểm soát trung tâm.

```
pawn_scores = [[0, 0, 0, 0, 0, 0, 0, 0],
               [50, 50, 50, 50, 50, 50, 50, 50],
               [10, 10, 20, 30, 30, 20, 10, 10],
               [5, 5, 10, 25, 25, 10, 5, 5],
               [0, 0, 0, 20, 20, 0, 0, 0],
               [5, -5, -10, 0, 0, -10, -5, 5],
               [5, 10, 10, -20, -20, 10, 10, 5],
               [0, 0, 0, 0, 0, 0, 0, 0]]
```

Mã (Knight)

Bảng điểm của quân Mã khuyến khích chiếm lĩnh các ô ở trung tâm (các ô `d4`, `e4`, `d5`, `e5`), nơi Mã có nhiều đường di chuyển hiệu quả nhất. Các vị trí ở biên và góc (`a1`, `h1`, `a8`, `h8`) bị trừ điểm mạnh vì Mã ở đó kém linh hoạt. Ngoài ra, những ô gần trung tâm như `b3`, `g3`, `b5`, `g5` cũng được cộng thêm chút điểm, hỗ trợ cho chiến thuật cắm chốt Mã. Ma trận điểm này thể hiện rõ tinh thần của Tartakover: "một bộ phận đứng lệch, cả ván đấu trở nên méo mó".

```
knight_scores = [[-50, -40, -30, -30, -30, -30, -40, -50],
                 [-40, -20, 0, 0, 0, 0, -20, -40],
                 [-30, 0, 10, 15, 15, 10, 0, -30],
                 [-30, 5, 15, 20, 20, 15, 5, -30],
                 [-30, 0, 15, 20, 20, 15, 0, -30],
                 [-30, 5, 10, 15, 15, 10, 5, -30],
                 [-40, -20, 0, 5, 5, 0, -20, -40],
                 [-50, -40, -30, -30, -30, -30, -40, -50]]
```

Tượng (Bishop)

Bảng điểm của quân Tượng tránh đặt ở các ô biên và góc, vì tầm hoạt động của Tượng bị hạn chế. Những ô như `b3`, `c4`, `b5`, `d3` được cộng thêm điểm để khuyến khích Tượng chiếm các đường chéo quan trọng. Các trung tâm kép (`d4`, `e5`...) có giá trị cao nhất, phản ánh sức

mạnh phối hợp với quân khác. Đồng thời, vị trí như `c3`, `f3` được thưởng thêm để cân bằng giá trị khi trao đổi với Mã.

```
bishop_scores = [[-20, -10, -10, -10, -10, -10, -10, -20],
                 [-10, 0, 0, 0, 0, 0, 0, -10],
                 [-10, 0, 5, 10, 10, 5, 0, -10],
                 [-10, 5, 5, 10, 10, 5, 5, -10],
                 [-10, 0, 10, 10, 10, 10, 0, -10],
                 [-10, 10, 10, 10, 10, 10, 10, -10],
                 [-10, 5, 0, 0, 0, 0, 5, -10],
                 [-20, -10, -10, -10, -10, -10, -10, -20]]
```

Xe (Rook)

Bảng điểm của quân Xe khuyến khích lên chiếm hàng bảy của đối phương, nơi Xe có thể tấn công Tốt lẻ. Các cột biên (`a`–`h`) bị trừ điểm để tránh Xe bị bó buộc phải bảo vệ Tốt bên cánh. Trung tâm hàng tám và hàng hai–sáu được đánh giá trung tính hoặc tích cực nhẹ, hỗ trợ cho việc di chuyển dọc cột mở. Ý tưởng "gerbil-like" (biến động linh hoạt) tập trung vào thế chiếm dọc và ngang hiệu quả.

```
rook_scores = [[0, 0, 0, 0, 0, 0, 0, 0],
               [5, 10, 10, 10, 10, 10, 10, 5],
               [-5, 0, 0, 0, 0, 0, 0, -5],
               [-5, 0, 0, 0, 0, 0, 0, -5],
               [-5, 0, 0, 0, 0, 0, 0, -5],
               [-5, 0, 0, 0, 0, 0, 0, -5],
               [-5, 0, 0, 0, 0, 0, 0, -5],
               [0, 0, 0, 5, 5, 0, 0, 0]]
```

Hậu (Queen)

Bảng điểm của quân Hậu đặt tiêu chí tránh góc và biên, vì Hậu ở đó dễ bị đối phương tấn công và kém linh hoạt. Các ô trung tâm (`d4`, `e4`, `d5`, `e5`) được cộng điểm nhẹ giúp Hậu có tầm hoạt động rộng. Một vài ô như `b3`, `c2` cũng có điểm thêm theo gợi ý, hỗ trợ chiến thuật tấn công đường chéo. Phần còn lại là tùy vào diễn biến tấn công–phòng ngự và các tính toán chiến thuật cụ thể.

```
queen_scores = [[-20, -10, -10, -5, -5, -10, -10, -20],
                [-10, 0, 0, 0, 0, 0, 0, -10],
                [-10, 0, 5, 5, 5, 5, 0, -10],
                [-5, 0, 5, 5, 5, 5, 0, -5],
                [0, 0, 5, 5, 5, 5, 0, -5],
                [-10, 5, 5, 5, 5, 5, 0, -10],
                [-10, 0, 5, 0, 0, 0, 0, -10],
                [-20, -10, -10, -5, -5, -10, -10, -20]]
```

Vua giữa trận (King – Middlegame)

Giữa trận, Vua cần đứng sau "hàng rào" Tốt, nên các ô trước lối nhảy Thành (`f2`–`h2`) được ưu tiên điểm nhẹ. Những ô sát cột (hàng bốn–sáu) đều bị trừ điểm mạnh để tránh Vua lộ diện. Vị trí lý tưởng là đằng sau Tốt nhưng không quá giữu trung tâm, đảm bảo khả năng chạy trốn khi đối thủ mở tấn công. Ý tưởng chính là giữ Vua an toàn, tận dụng bức tường là các quân Tốt.

```
king_middle_score = [[-30, -40, -40, -50, -50, -40, -40, -30],
                     [-30, -40, -40, -50, -50, -40, -40, -30],
                     [-30, -40, -40, -50, -50, -40, -40, -30],
                     [-30, -40, -40, -50, -50, -40, -40, -30],
                     [-20, -30, -30, -40, -40, -30, -30, -20],
                     [-10, -20, -20, -20, -20, -20, -20, -10],
                     [20, 20, 0, 0, 0, 0, 20, 20],
                     [20, 30, 10, 0, 0, 10, 30, 20]]
```

Vua cuối trận (King – Endgame)

Cuối trận, khi ít quân, Vua trở thành quân tấn công, nên các ô trung tâm (`d4`, `e4`, `d5`, `e5`) được thưởng điểm cao nhất. Các ô biên và góc vẫn bị trừ điểm vì Vua ở đó kém linh hoạt. Hàng hai–ba gần trung tâm có điểm nhẹ để Vua có thể hỗ trợ chiếm trung tâm và ép góc đối phương. Sự chuyển đổi từ "ẩn mình" sang "xông pha" phản ánh khác biệt rõ giữa giữa trận và cuối trận.

```
king_end_score = [[-50, -40, -30, -20, -20, -30, -40, -50],
                  [-30, -20, -10, 0, 0, -10, -20, -30],
                  [-30, -10, 20, 30, 30, 20, -10, -30],
                  [-30, -10, 30, 40, 40, 30, -10, -30],
                  [-30, -10, 30, 40, 40, 30, -10, -30],
                  [-30, -10, 20, 30, 30, 20, -10, -30],
                  [-30, -30, 0, 0, 0, 0, -30, -30],
                  [-50, -30, -30, -30, -30, -30, -30, -50]]
```

2.2.3 Các hàm đánh giá trạng thái bàn cờ

getMaterialScore(gs)

Hàm này duyệt qua tất cả các ô trên `gs.board` để tính tổng giá trị của mỗi bên dựa trên số lượng và bản chất của quân cờ. Nếu ô hiện tại chứa quân trắng, giá trị tương ứng từ `piece_score` được cộng vào `white_score`; ngược lại, nếu là quân đen thì cộng vào `black_score`. Sau khi quét xong, hàm trả về hiệu số `black_score - white_score`. Giá trị trả về phản ánh ưu thế về lực lượng thuần túy, không xét đến vị trí hay cấu trúc bàn cờ.

```python
# Heuristic 1
Tabnine | Edit | Test | Explain | Document
@staticmethod
def getMaterialScore(gs):
    white_score = 0
    black_score = 0
    for row in range(len(gs.board)):
        for col in range(len(gs.board[row])):
            piece = gs.board[row][col]
            if piece != "--":
                if piece[0] == "W":
                    white_score += piece_score[piece[1]]
                else:
                    black_score += piece_score[piece[1]]
    return black_score - white_score
```

getPiecePositionScore(self, gs)

Hàm này đánh giá ưu thế không gian dựa trên ma trận `piece_position_scores`. Trước tiên, dựa vào biến cờ `isEndGame` để lựa chọn ma trận cho Vua giữa trận hoặc cuối trận. Sau đó, với mỗi ô chứa quân, nó cộng hoặc trừ điểm theo ma trận (cộng điểm cho quân đen, trừ cho quân trắng), để thu về tổng chênh lệch vị thế dựa trên vị trí chiến lược. Trả về kết quả giá trị nguyên dương nếu Đen chiếm ưu thế, âm nếu Trắng chiếm ưu thế.

```python
# Heuristic 2
Tabnine | Edit | Test | Explain | Document
def getPiecePositionScore(self, gs):
    score = 0
    if self.isEndGame:
        piece_position_scores['WK'] = king_end_score
        piece_position_scores['BK'] = king_end_score[::-1]
    else:
        piece_position_scores['WK'] = king_middle_score
        piece_position_scores['BK'] = king_middle_score[::-1]

    for row in range(len(gs.board)):
        for col in range(len(gs.board[row])):
            piece = gs.board[row][col]
            if piece != "--":
                if piece[0] == "B":
                    score += piece_position_scores[piece][row][col]
                else:
                    score -= piece_position_scores[piece][row][col]
    return score
```

evaluation(self, gs)

Hàm này tổng hợp điểm Material và điểm Position để cho ra "điểm" cuối cùng của thế cờ. Đầu tiên, nó gọi `__checkEndGame` để cập nhật giai đoạn trận (giữa hoặc cuối). Nếu lượt AI mà không có nước đi hợp lệ, trả về `-20000` để báo tình huống chiếu bí. Nếu AI đóng vai quân Đen, giá trị đánh giá là `getPiecePositionScore + getMaterialScore`; nếu AI là Trắng, lấy phủ định của hai thành phần này. Điểm trả về được sử dụng làm giá trị để đánh giá trong thuật toán tìm kiếm (Alpha–Beta).

```python
Tabnine | Edit | Test | Explain | Document
def __checkEndGame(self,gs):
    if gs.piece_ingame['WQ'] == 0 and gs.piece_ingame['BQ'] == 0:
        return True
    if gs.piece_ingame['WQ'] == 1 and gs.piece_ingame['BQ'] == 1:
        white_minor_piece = 0
        black_minor_piece = 0
        for u, v in gs.piece_ingame.items():
            if u[1] == 'N' or u[1] == 'B':
                if u[0] == 'W':
                    white_minor_piece += v
                else:
                    black_minor_piece += v
        if white_minor_piece <= 1 and black_minor_piece <= 1:
            return True

    return False
Tabnine | Edit | Test | Explain | Document
def evaluation(self, gs):
    self.__checkEndGame(gs)
    if gs.getTurn()==self.aiTurn:
        if len(gs.getValidMoves())==0: return -20000
    if self.aiTurn == 'B':
        score = self.getPiecePositionScore(gs) + AIEngine.getMaterialScore(gs)
    else:
        score = - self.getPiecePositionScore(gs) - AIEngine.getMaterialScore(gs)
    return score
```

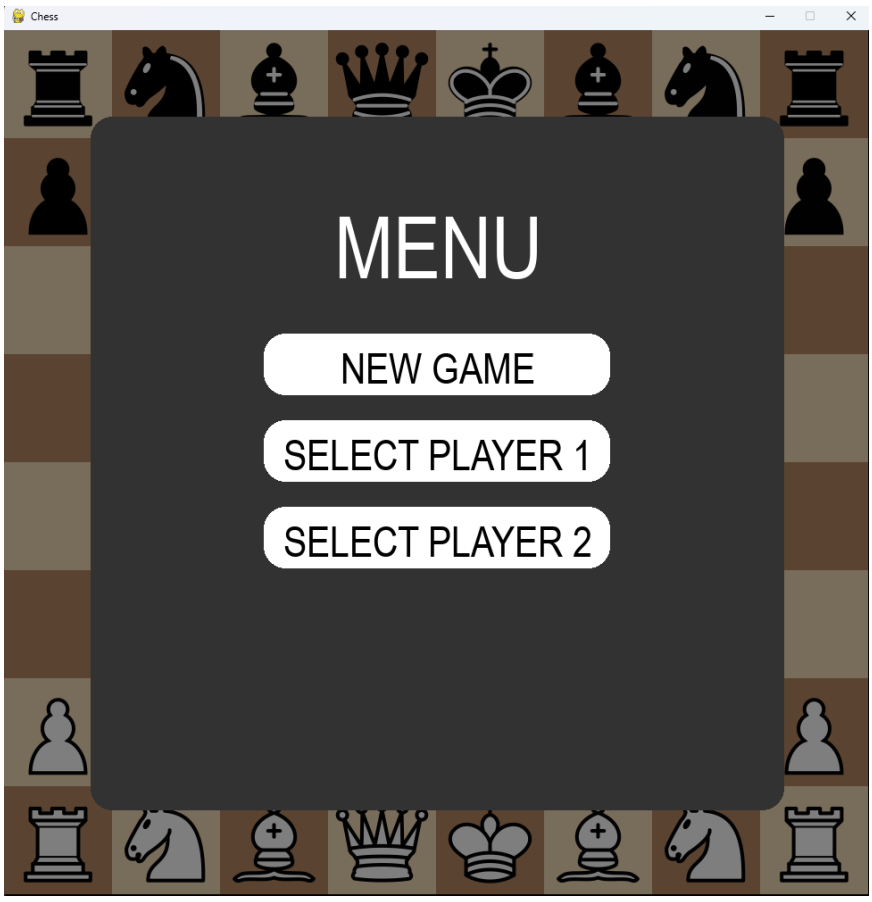
3 Thực hiện, xây dựng chương trình

3.1 Mô tả chương trình

- Chương trình được xây dựng trên môi trường Windows.
- Ngôn ngữ lập trình sử dụng: Python.
- Mã nguồn được lưu trữ trên GitHub: https://github.com/sammiesamm/A2_TTNT
- Hướng dẫn cách chạy chương trình ở file `README.md`.
- Các thư viện sử dụng: pygame (version 2.6.1), random, threading, copy, os.

3.2 Giao diện chương trình

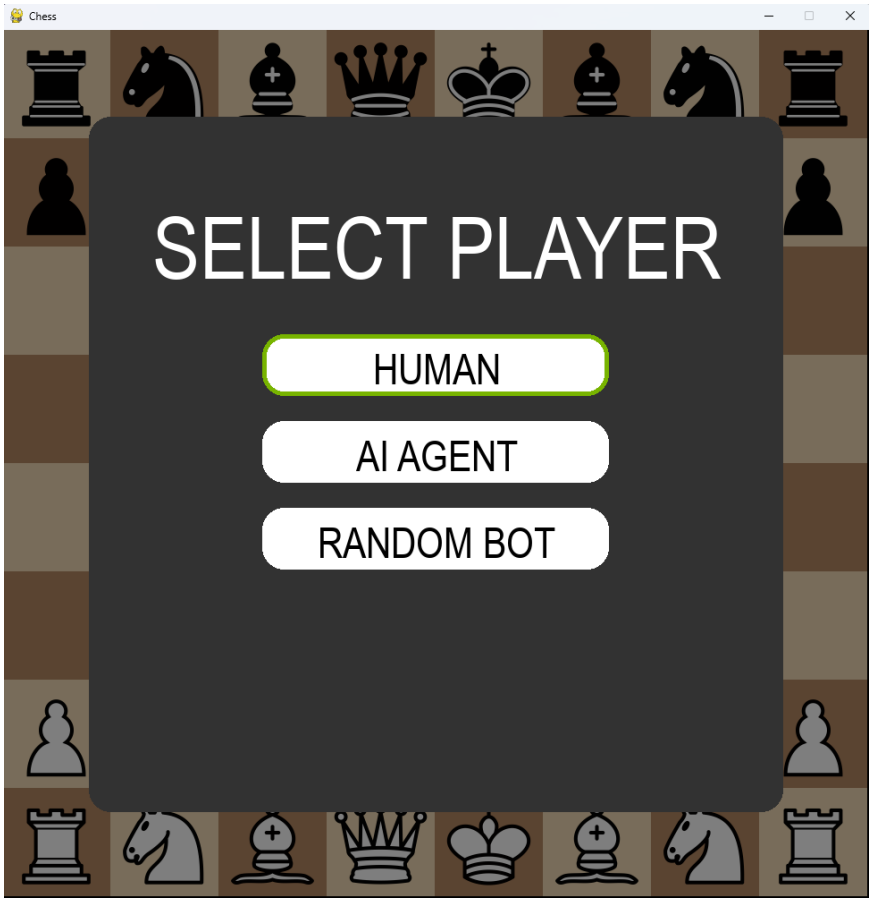
3.2.1 Menu



Giao diện menu gồm ba mục chính như trên lần lượt là:

- New game: tạo trò chơi mới

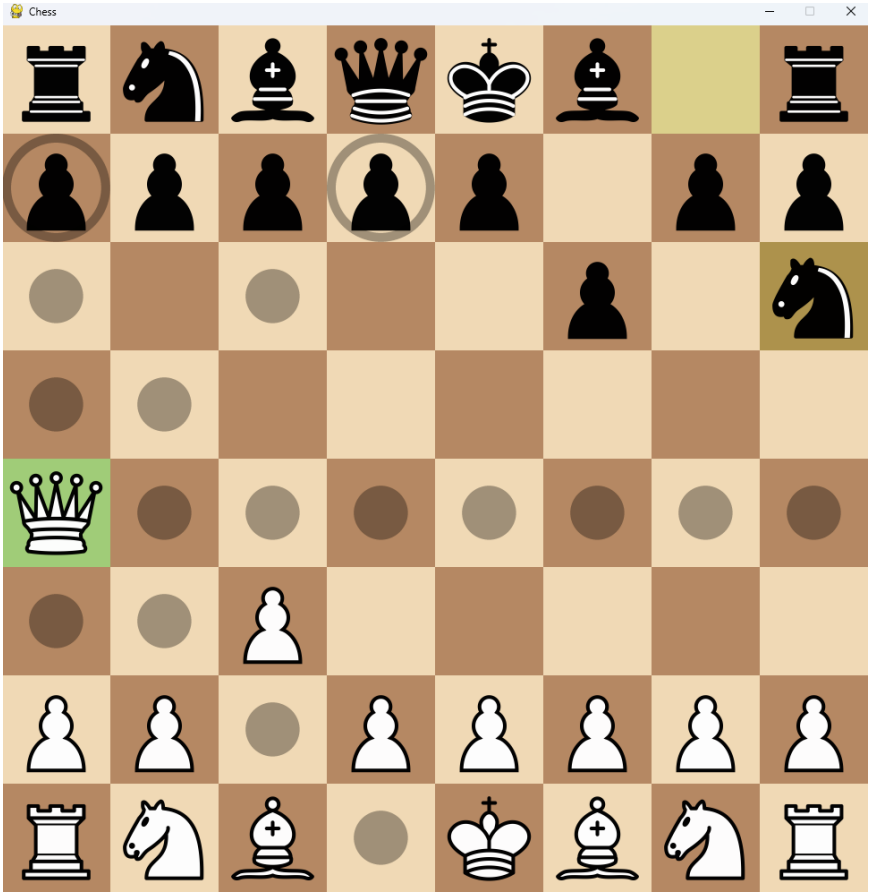
- Select player 1: chọn người chơi cho quân trắng
- Select player 2: chọn người chơi cho quân đen



Giao diện chọn người chơi có các mục sau:

- Human: Người điều khiển các quân cờ
- AI agent: AI điều khiển các quân cờ với ba chế độ là dễ (Easy), vừa (Medium) và khó (Hard)
- Random bot: người chơi lựa chọn ngẫu nhiên các quân cờ

3.2.2 Điều khiển trò chơi



- Thực hiện nước đi: Đối với người điều khiển trò chơi là con người, khi đến lượt của người chơi thì chọn vào quân cờ để hiển thị các nước đi hợp lệ, chọn tiếp vào các vị trí hợp lệ để thực hiện nước đi. Với người chơi không phải là người thì tự động tính toán và đưa ra nước đi hợp lệ

- Các phím điều khiển: Ấn phím ESC để bật menu và click chuột để chọn chế độ. Ấn phím Ctrl+Z để quay trở lại các bước trước đó.

3.3 Mô hình hóa bài toán

3.3.1 Biểu diễn bàn cờ

Ma trận 8×8

Bàn cờ được lưu trong biến **self.board**: một danh sách hai chiều kích thước 8×8. Mỗi ô trên bàn cờ là một chuỗi gồm hai ký tự:

- Ký tự đầu tiên (`W` hoặc `B`) biểu thị màu quân: trắng hoặc đen.

- Ký tự thứ hai biểu thị loại quân:
  - `P`: Tốt (Pawn)
  - `N`: Mã (Knight)
  - `B`: Tượng (Bishop)
  - `R`: Xe (Rook)
  - `Q`: Hậu (Queen)
  - `K`: Vua (King)
- Các ô trống được ký hiệu bằng chuỗi `"--"`.

Theo dõi trạng thái bàn cờ

- **self.turn:** màu quân đang di chuyển hiện tại (`W` hoặc `B`).
- **self.kingLocation:** vị trí (hàng, cột) của vua mỗi bên để hỗ trợ kiểm tra chiếu nhanh chóng.
- **self.current_castling_rights:** lưu bốn cờ trạng thái (wks, wqs, bks, bqs) biểu thị quyền nhập thành:
  - wks: Trắng còn quyền nhập thành cánh vua (kingside)
  - wqs: Trắng còn quyền nhập thành cánh hậu (queenside)
  - bks: Đen còn quyền nhập thành cánh vua (kingside)
  - bqs: Đen còn quyền nhập thành cánh hậu (queenside)
- **self.enpassant_possible:** tọa độ ô có thể bị ăn qua đường (en passant).
- **self.moveLog:** danh sách các nước đi đã thực hiện (dùng để undo).
- **self.piece_ingame:** số lượng còn lại của từng loại quân (dùng để phát hiện các trường hợp kết thúc ván cờ do thiếu chất).

Quy ước tọa độ hàng và cột

- Hàng 0 tương ứng với phía quân đen; hàng 7 tương ứng với phía quân trắng.
- Các cột 0 đến 7 tương ứng với các cột từ a đến h.

Khi cần chuyển đổi giữa tọa độ nội bộ và ký hiệu chuẩn cờ Tế (ví dụ: từ $(6, 4)$ thành `e2`), sử dụng hàm ánh xạ được định nghĩa trong phương thức `Move.getChessNotation()`.

3.3.2 Sinh nước đi hợp lệ

Quy trình sinh nước đi hợp lệ gồm hai bước chính:

1. Tạo tất cả các nước đi hợp lệ về mặt cấu trúc (*pseudo-legal moves*).
2. Lọc các nước đi theo chiếu và ghim (*pins/checks*).

Xác định vị trí chiếu và ghim

Hàm `_getPinAndCheckPieces()` quét 8 hướng xung quanh vua để tìm:

- **Checks**: các quân đối phương đang chiếu trực tiếp vua (trên các đường thẳng hoặc đường chéo).
- **Pins**: các quân cùng màu bị ghim trên đường thẳng hoặc chéo giữa vua và quân đối phương, chỉ được phép di chuyển dọc theo đường đó.

Ngoài ra, hàm cũng kiểm tra chiếu bởi mã (knight) thông qua 8 vector di chuyển đặc biệt của quân mã.

Sinh pseudo-legal moves

Với từng loại quân, cách sinh nước đi cụ thể như sau:

- **Pawn** (`_getPawnMoves`):
  - Tiến 1 ô, hoặc 2 ô nếu là lần di chuyển đầu tiên.
  - Ăn chéo quân đối phương.
  - Ăn en passant.
  - Phong cấp (promotion) khi tới hàng cuối.
  - Sử dụng cờ `attackAble` để phân biệt giữa sinh ô tấn công và ô di chuyển (hữu ích trong kiểm tra chiếu).
- **Knight** (`_getKnightMoves`):
  - Sinh 8 nước nhảy theo hình chữ L.
  - Không quan tâm chắn đường (knight có thể "nhảy" qua các quân khác).
- **Bishop / Rook / Queen** (`_getBishopMoves`, `_getRookMoves`, `_getQueenMoves`):
  - Quét theo các hướng tương ứng (chéo cho Bishop, thẳng cho Rook, cả hai cho Queen).
  - Dừng lại khi gặp quân cùng màu.

  - Nếu gặp quân đối phương, thêm nước ăn quân đó rồi kết thúc quét theo hướng đó.

- **King** (`_getKingMoves`):
  - Tương tự như Rook/Bishop nhưng chỉ di chuyển đúng 1 ô theo mỗi hướng.

Kết quả của bước này là danh sách các *pseudo-legal moves* — tức các nước đi hợp lệ về mặt cấu trúc, nhưng chưa xét đến chiếu hoặc ghim.

Các nước đi bao gồm:

- Nước ăn quân
- Nước tiến
- Các nước đặc biệt như:
  - Đột phá 2 ô của tốt (pawn) khi xuất phát
  - Ăn en passant
  - Phong cấp (promotion)
  - Nhập thành (castling)

Các hàm này chỉ xét va chạm (`_checkCollision`) hoặc giới hạn tấn công (`attackAble`), chưa xét đến chiếu.

Lọc theo chiếu và ghim

- **Nếu không bị chiếu (no checks)**:
  - Với quân bị ghim: chỉ giữ các nước đi mà ô đích nằm trên đường ghim.
  - Với quân không bị ghim: giữ toàn bộ pseudo-legal moves.
- **Nếu bị chiếu một nước (single check)**:
  - Cho phép:
    * Nước di chuyển bắt quân chiếu (capture checker).
    * Nước đẩy quân chắn vào giữa (block đường chiếu), nếu quân chiếu là Rook/Bishop/Queen.
- **Nếu bị chiếu hai nước (double check)**:
  - Chỉ cho phép di chuyển vua để thoát chiếu.

Castling

Sau khi sinh xong pseudo-legal moves, gọi `getCastleMoves()` nếu thỏa mãn:

- Các ô vua đi qua và ô đích không bị tấn công.
- Cả vua lẫn xe liên quan đều chưa di chuyển (dựa trên `current_castling_rights`).

Kết quả

Kết quả trả về là danh sách `moves` gồm các đối tượng `Move` hợp lệ, đã vượt qua tất cả các kiểm tra về chiếu và ghim, sẵn sàng để đánh giá thứ tự ưu tiên hoặc sử dụng trong thuật toán tìm kiếm như minimax.

3.3.3 Chiến lược tìm kiếm

Thay vì đi thẳng đến một độ sâu cố định, ta "lặp" tìm kiếm từ sâu 1, 2, 3... lên đến max_depth. Việc tìm kiếm ở độ sâu nông giúp thu thập thông tin về nước đi khả dĩ tốt. Khi chuyển sang độ sâu lớn hơn, ta có thể thử những nước đi hứa hẹn trước, điều này làm cắt tỉa Alpha-Beta hiệu quả hơn (cắt nhiều hơn, nhanh hơn).

Mỗi lần hoàn thành một độ sâu, ta có một "nước đi tạm tốt nhất" nếu bị dừng giữa chừng (nhờ giới hạn thời gian). Trong nhiều ứng dụng (như cờ vua máy tính), AI phải trả lời trong một khoảng thời gian cố định (ví dụ 2 giây mỗi nước). Nếu chỉ chạy Minimax một lần với độ sâu lớn, khi hết giờ hàm có thể chưa kịp hoàn thành và ta mất luôn nước đi. việc tìm kiếm theo độ sau từ nhỏ đến lớn đảm bảo rằng sau mỗi "lần nhỏ" (depth = 1, 2, 3...) ta luôn có một "nước đi tạm tốt nhất" để dùng nếu bị ngắt giữa chừng.

Kết hợp với Alpha-Beta, chiến lược tìm kiếm này thường cho hiệu quả cắt tỉa tốt hơn vì các bước nông xác định thứ tự nước đi cho bước sâu hơn. So với các chiến lược khác như MCTS, chiến lược tìm kiếm này chỉ cần duy trì ngăn xếp đệ quy mà không tốn thêm cấu trúc phức tạp.

3.4 Thuyết trình và Kiểm thử

Video trình bày về dự án cờ vua và kiểm thử được đăng tải ở link Youtube sau: https://youtu.be/y0_HdX3Okxg

4 Tài liệu

[1] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson Education, 2020.

[2] Claude E. Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine*.

[3] "Chess Programming Wiki," https://www.chessprogramming.org.

[4] Tord Romstad, Marco Costalba, Joona Kiiski, và Gary Linscott, "Stockfish Chess Engine", https://github.com/official-stockfish/Stockfish.

[5] "Simplified Evaluation Function", https://www.chessprogramming.org/Simplified_Evaluation_Function.