Mục lục

CHƯƠNG 1. KHẢO SÁT YÊU CẦU BÀI TOÁN

1.1. Mô tả yêu cầu bài toán

Tóm tắt bài toán: Viết chương trình cho phép chơi cờ vua với máy tính

Yêu cầu:

- Các nước đi trong bàn cờ đúng với các luật trong cờ vua
- Chương trình có đồ họa
- Máy tính phải phản ứng các nước cờ nhanh (~3s)

Thách thức:

- Hệ số phân nhánh lớn
- Quản lý các nước đi của một trạng thái
- Làm quen với Python

Lợi thế:

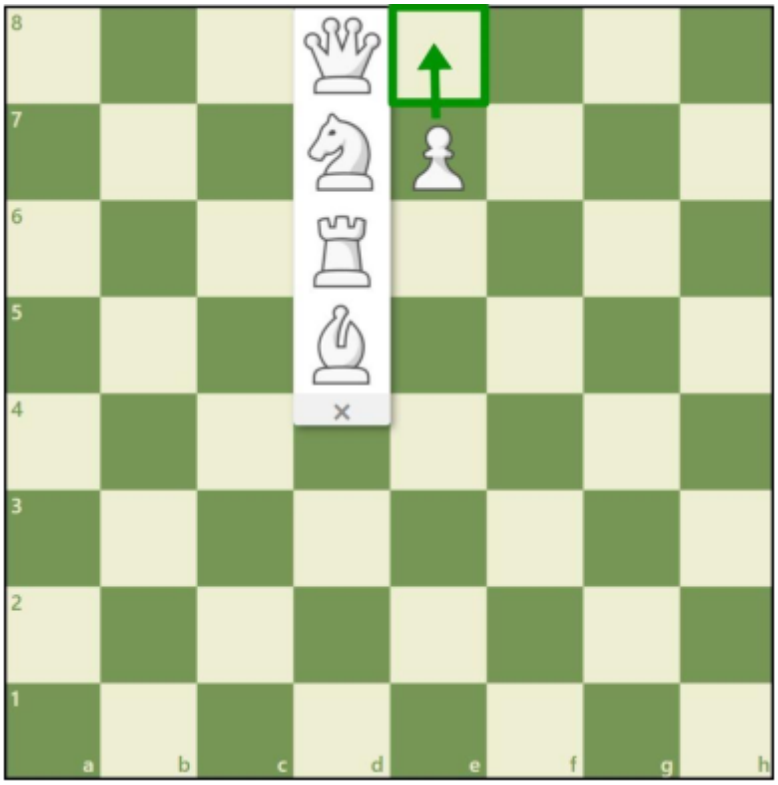
- Thư viện hỗ trợ xây dựng đồ họa: PySimpleGUI
- Thư viện hỗ trợ kiểm soát các trạng thái của bàn cờ: chess

1.2. Các luật di chuyển đặc biệt trong cờ vua

Ngoài các luật và các nước đi cơ bản như tượng đi chéo, xe đi ngang và dọc, tốt đi thẳng ăn chéo… thì trong cờ vua còn có một vài luật đặc biệt không theo các quy tắc cơ bản mà người chơi có thể không biết hoặc còn nhầm lẫn.

1.2.1. Phong cấp quân tốt

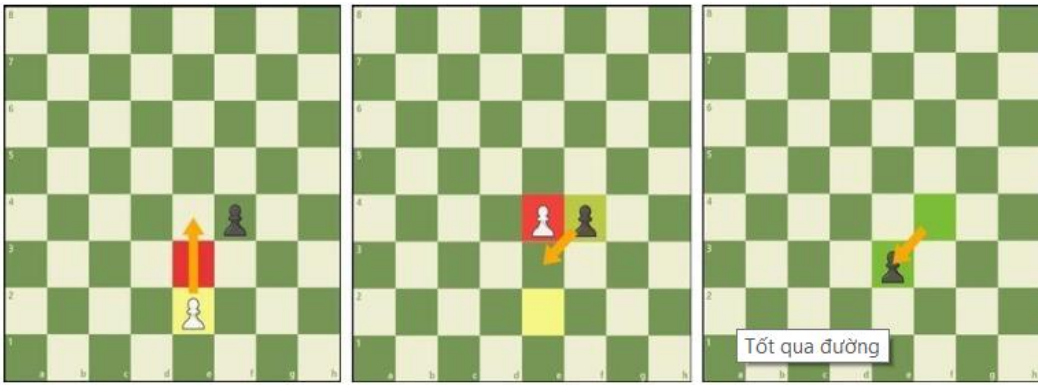
Quân tốt có một khả năng đặc biệt là nếu nó di chuyển đến hàng cuối cùng phía bên kia bàn cờ, nó sẽ được thay thế bằng một quân cờ khác (đây gọi là Phong cấp).



Tốt có thể được phong cấp thành bất cứ quân nào trong 4 quân: hậu, mã, xe, tượng. Có một hiểu nhầm khá phổ biến là tốt chỉ có thể thay thế thành quân cờ đã bị bắt trước đó. Điều đó hoàn toàn KHÔNG đúng. Tốt thường được phong cấp thành quân hậu. Chỉ quân tốt mới có thể được phong cấp.

1.2.2. Ăn tốt qua đường

Luật chơi cuối cùng áp dụng cho quân tốt là "en passant", là tiếng Pháp có nghĩa "bắt tốt qua đường". Nếu quân tốt xuất phát với việc di chuyển hai ô đầu tiên, và sau đó kết thúc tại vị trí đứng cạnh quân tốt đối phương (nhảy qua khỏi ô mà tốt đối phương có thể bắt nó), con tốt đối phương lúc này có quyền được bắt quân tốt đó.



Cách di chuyển đặc biệt này chỉ có thể sử dụng ngay sau khi quân tốt thực hiện bước đi đầu tiên, sang lượt sau sẽ không thể bắt nữa.

1.2.3. Nhập thành

Có một luật đặc biệt trong cờ vua được gọi là nhập thành. Nước đi này cho phép bạn thực hiện 2 việc rất quan trọng trong cùng một lượt: đưa quân Vua của bạn tới vị trí an toàn hơn (tùy trường hợp), và cho phép quân Xe rời khỏi góc bàn cờ và tham gia cuộc chơi. Lượt đi nhập thành của một người chơi bao gồm: di chuyển Vua hai ô về một phía của bàn cờ (chỉ được phép đi ngang) và quân Xe (trong góc, hướng Vua di chuyển) ra phía ngoài, đứng cạnh Vua nhưng ở phía ngược lại. Tuy nhiên, phải đảm bảo các yêu cầu sau thì mới đi được nước nhập thành:

- Lượt này phải là lượt đi đầu tiên của vua
- Lượt này phải là lượt đi đầu tiên của quân xe
- Không được có bất kỳ quân nào khác đứng giữa vua và xe
- Vua không thể bị chiếu hoặc đi cắt qua đường chiếu

Khi nhập thành hướng quân Vua gần với mép bàn cờ hơn được gọi là nhập thành "bên Vua". Nhập thành hướng ngược lại, hướng mà quân hậu đứng, được gọi là nhập thành "bên Hậu". Dù hướng nào đi nữa, Vua chỉ được phép đi 2 ô khi nhập thành.

CHƯƠNG 2. PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN

2.1. Các thư viện hỗ trợ

2.1.1. Thư viện PySimpleGUI

Thư viện PySimpleGUI giúp xây dựng đồ họa cho chương trình.

2.1.2. Thư viện python-chess

Thư viện python-chess giúp chương trình dễ dàng quản lý trạng thái các nước cờ, cung cấp các chức năng hữu ích như:

- Hiển thị trạng thái bàn cờ
- Sinh ra các nước đi hợp lệ cho một trạng thái
- Cập nhật trạng thái sau các nước đi

2.2. Thuật toán sử dụng

Để có thể tự động thực hiện các nước cờ đối đầu với người chơi, chương trình đã sử dụng thuật toán minimax cùng với các thuật toán khác với mục đích cải thiện tốc độ cho thuật toán minimax:

2.2.1. Minimax

Ý tưởng:

- Là giải thuật đệ quy lựa chọn bước đi kế tiếp cho một trạng thái của bàn cờ dựa trên đánh giá heuristic.
- Điểm heuristic của một trạng thái là tổng điểm của các quân cờ và vị trí của quân cờ đó trên bàn cờ.

Mục đích: Một chiến lược tối ưu là một chuỗi các nước đi giúp đưa đến trạng thái đích mong muốn (vd: chiến thắng).

Thực hiện:

- Chiến lược của MAX bị ảnh hưởng (phụ thuộc) vào các nước đi của MIN và ngược lại
- MAX cần chọn một chiến lược giúp cực đại hóa giá trị hàm mục tiêu với giả sử

là MIN đi các nước đi tối ưu, MIN cần chọn một chiến lược giúp cực tiểu hóa giá trị hàm mục tiêu

- Chiến lược này được xác định bằng việc xét giá trị MINIMAX đối với mỗi nút trong cây biểu diễn trò chơi

Sau đây sẽ là các thuật toán sử dụng để tối ưu tốc độ của giải thuật minimax:

2.2.2. Alpha-Beta Pruning

Ý tưởng: Nếu một nhánh tìm kiếm nào đó không thể cải thiện đối với giá trị mà chúng ta đã có, thì không cần xét đến nhánh tìm kiếm đó nữa.

Mục đích: Tăng hiệu quả của chương trình mà không làm ảnh hưởng đến kết quả.

Thực hiện:

- Alpha là giá trị của nước đi tốt nhất đối với MAX (giá trị tối đa) tính đến hiện tại đối với nhánh tìm kiếm
- Nếu v là giá trị tồi hơn alpha, MAX sẽ bỏ qua nước đi ứng với v
- Beta được định nghĩa tương tự đối với MIN

Hiệu quả thuật toán: Thuật toán cắt tỉa alpha-beta là thuật toán cơ bản để tối ưu thuật toán, là nền cho các thuật toán cải thiện khác.

2.2.3. Sắp xếp điểm các nước đi trước khi thực hiện

Ý tưởng: Các nước đi sẽ được sắp xếp lại khi thực hiện đào sâu. Và được sắp xếp dựa trên số điểm heuristic thay đổi nếu đi nước cờ đó (tương đương với việc tính điểm của nước cờ đó với độ sâu là 1).

Mục đích: Đưa các nước "có vẻ" tốt hơn lên trước, từ đó cắt tỉa được nhiều nhánh hơn.

Thực hiện: Để có thể sắp xếp các nước đi dựa trên điểm số thay đổi được, chương trình sử dụng hàm calculate_score(move : chess.Move) sử dụng để tính sự thay đổi của điểm heuristic khi di chuyển nước đi move.

Hiệu quả thuật toán: giảm thêm 50%~60% thời gian thực hiện so với chỉ dùng những thuật toán đã nêu (cắt tỉa alpha-beta).

2.2.4. Zobrist hashing:

Ý tưởng: Thuật toán tìm một giá trị Hash (64-bits) đại diện cho một trạng thái của bàn cờ.

Mục đích: Zobrist hashing hỗ trợ để triển khai các thuật toán cần lưu lại các kết quả của một trạng thái.

Thực hiện:

- Sử dụng một mảng 2 chiều kích cỡ 12x64 (12 quân cờ, 64 vị trí), mỗi phần tử của mảng là một giá trị random 64 bits.
- Khởi tạo mảng và gán các giá trị random được thực hiện duy nhất 1 lần ngay sau khi khởi chạy chương trình.
- Thực hiện các hàm xor (^) để tính giá trị hash cho một trạng thái.
  - o Ví dụ: Nếu có quân xe đen (mã số là 3) đứng ở ô h3 (ô số 23) thì kết quả hash là hash ^= table[3][23]
- Vì tính chất của hàm xor, khi xor với chính nó sẽ bằng 0 và khi xor với 0 thì không thay đổi nên ta có thể tính mã hash cho một nước cờ chứ không cần thiết phải tính lại các quân cờ trên 64 ô cờ:
  - o Khi di chuyển một quân cờ (mã là 1) từ ô 25 sang ô 56, chúng ta chỉ cần thay đổi: hash ^= table[1][25]; hash^=table[1][56]

Hiệu quả thuật toán: Zobrist hashing không tác động trực tiếp đến tốc độ của chương trình, tuy nhiên nó giúp các thuật toán cần lưu trạng thái bàn cờ tiết kiệm thời gian và bộ nhớ hơn thay vì phải lưu trạng thái bàn cờ dựa trên chuỗi fen (tốn thời gian sinh chuỗi, tốn bộ nhớ).

Pseudocode:

```
table[12][64]
FOR i = 1 TO 12
    FOR j = 1 TO 64
        table[i][j] = random(số 64-bits)
```

2.2.5. Bảng chuyển vị (Tranposition table)

Ý tưởng: Lưu lại cặp (điểm, độ sâu) của các trạng thái được duyệt.

Mục đích: Tránh lặp lại các thao tác mà mình đã làm trước đó.

Thực hiện:

- Nếu trạng thái đang xét chưa có trong bảng chuyển vị: Thực hiện đào sâu như bình thường, sau đó lưu điểm cùng với độ sâu vừa đào vào bảng chuyển vị
- Nếu trạng thái đang xét đã có trong bảng chuyển vị:
  - Nếu độ sâu đã lưu >= độ sâu cần đào: Lấy giá trị điểm có sẵn trong bảng chuyển vị thay vì thực hiện đào sâu để tính điểm
  - Nếu độ sâu đã lưu nhỏ hơn độ sâu cần đào: Coi như trạng thái đó chưa có trong bảng chuyển vị, thực hiện như bình thường
- Các giá trị được lưu lại bằng cách sử dụng cấu trúc dữ liệu map với cặp key, value là:
  - Key: là Zobrish hash của trạng thái cần lưu
  - Value: là cặp giá trị (điểm, độ sâu)

Hiêu quả thuật toán: Giảm thêm ~25% so với chỉ sử dụng các thuật toán trước

2.2.6. Principal Variation Search (PV-search)

Ý tưởng: Lưu lại các nước cờ phản ứng tốt cho một trạng thái mà mình đã duyệt. Nước cờ phản ứng tốt là nước cờ đạt được lợi nhuận tốt nhất cho bên thực hiện.

Mục đích: Duyệt các nước cờ có khả năng là tốt nhất trước, từ đó tăng hiệu quả của thuật toán cắt tỉa alpha-beta.

Thực hiện:

- Lưu lại các nước đi vượt qua ngưỡng beta (đối với min) hoặc vượt qua ngưỡng alpha (đối với max) của một trạng thái vào Map.
- Các pv-moves sẽ được tìm kiếm đầu tiên khi đến trạng thái nếu trạng thái đó đã được lưu các pv-moves trước đó.

Hiệu quả thuật toán: Giảm thêm khoảng ~25% thời gian chạy so với chỉ sử dụng các thuật toán đã nêu trước.

Pseudocode cho thuật toán minimax + sắp xếp điểm các nước đi trước khi thực hiện + bảng chuyển vị + PV-search:

```
MINIMAX(depth, alpha, beta, isMaxPlayer)
BEGIN
    IF không còn nước đi THEN
        RETURN: -∞: đen thắng, ∞: trắng thắng, 0: hòa
    IF depth = MAX_DEPTH THEN
        RETURN score(trạng thái)
    IF trạng thái ∈ transposition_table AND
    độ sâu tìm kiếm trc đó >= MAX_DEPTH-depth THEN
        RETURN điểm đã tìm kiếm
    move_list = Các nước đi hợp pháp của trạng thái
    better_move = sort(move_list)
    IF có pv-nodes THEN better_move = pv-nodes + better_move
    FOR move IN better_move:
        BEGIN
            Score = MINIMAX(depth+1, alpha, beta)
            IF isMaxPlayer = true THEN
                IF score < beta THEN
```

```
                    beta = score
                    Thêm move vào pv_move
                IF score <= alpha THEN
                    RETURN alpha
            ELSE
                IF score > alpha THEN
                    alpha = score
                    Thêm move vào pv_move
                IF score >= beta THEN
                    RETURN alpha
        END
    Đảo ngược pv_move
    IF isMaxPlayer = true THEN
        tranpos_table[present_hash] = (alpha, MAXDEPTH–depth)
    ELSE
        tranpos_table[present_hash] = (beta, MAXDEPTH–depth)
END
```

2.2.7. Đào sâu lặp lại (Iterative deepening)

Ý tưởng: Tăng độ sâu dần dần bắt đầu từ 2 (vì khi sắp xếp các nước đi thì cũng đã là đào sâu 1).

Mục đích: Để có được pv-moves cho một trạng thái qua các độ sâu tăng dần, ví dụ như khi duyệt độ sâu n thì các pv-moves của độ sâu n-1 (các nước đi tốt nhất nếu đào độ sâu n-1) sẽ được tìm kiếm trước.

Thực hiện:

- Tăng dần MAX_DEPTH bắt đầu từ 2.
- Với mỗi lần tăng thì thực hiện đào sâu với độ sâu là MAX_DEPTH

Hiệu quả thuật toán: Đào sâu lặp lại cải thiện thuật toán pv-search.

Pseudocode:

```
ITERATIVE_DEEPENING():
BEGIN
    FOR depth = 2 to 4
        MAX_DEPTH = depth
        best_move=minimax(0,-800011,800011,isMaxPlayer=False)
    RETURN best_move
END
```

2.2.8. Xử lý cuối game

Khi trên bàn cờ còn lại ít quân cờ thì hàm tính giá trị heuristic cần phải thay đổi. Thay vì tính giá trị của quân cờ dựa vào một bảng chấm điểm vị trí như cũ thì:

- Cần ưu tiên các quân vua đi ra giữa bàn cờ
- Ưu tiên các quân cờ khác tiến đến gần vua của đối phương

 Bên thắng sẽ có xu hướng tiếp cận và dồn quân vua của địch vào góc bàn cờ, tiến gần đến chiến thắng

CHƯƠNG 3. THIẾT KẾ CỦA CHƯƠNG TRÌNH

3.1. Kết quả chương trình minh họa

Sau quá trình phát triển nhóm đã có được kết quả như ý muốn. Sản phẩm được hoàn thành là chương trình được code bằng python cho phép chơi cờ vua với máy tính, đạt được các mục tiêu mà nhóm đề ra như đồ họa, thời gian chạy đủ nhanh…

Một vài thông tin về chương trình:

- Độ sâu chương trình thực hiện là 4
- Thời gian chạy trong khoảng 1 đến 3s, các nước đi đầu tiên sẽ thực hiện lâu hơn

3.2. Cấu trúc chương trình

Chương trình gồm 2 file là config.py và covua.py

3.2.1. File config.py

File config.py chứa các dữ liệu sử dụng cho chương trình:

- Mảng piece[]: dùng để chuyển từ ký hiệu quân cờ thành các số từ 1 đến 12.
- Mảng piece_score[]: chứa điểm của các quân cờ dùng để đánh giá điểm heuristic
- Mảng piece_position_score[]: chứa điểm để đánh giá vị trí của quân cờ.

3.2.2. File covua.py

File covua.py bao gồm 3 class: class GUI, class Game, class Bot

Class GUI: bao gồm các hàm để tạo đồ họa cho chương trình

- Hàm render_square(image, key, location): để sinh ra đồ họa cho 64 ô của bàn cờ với image là hình ảnh quân cờ, key là key của event khi người dùng click vào, location là vị trí của ô cờ.
- Hàm create_board_layout(): sử dụng để tạo layout cho bàn cờ.
- Hàm update_board(): sử dụng để cập nhật lại đồ họa cho vị trí của các quân cờ.
- Hàm change_square_selected(position): đổi màu ô position thành màu của ô được chọn.
- Hàm restore_square_color(position): thay đổi màu của ô tại vị trí position thành

màu gốc.

- Hàm choose_piece_promotion(): hiển thị cửa sổ cho phép chọn quân cờ để phong cấp.

- Hàm message(message): hiển thị cửa sổ thông báo messsage (thắng, thua hoặc hòa).

Class Game: bao gồm các hàm dùng để vận hành game

- Hàm calc_piece_quantity(): trả về giá trị là số quân cờ còn lại trên bàn cờ.

- Hàm play(event): là hàm nhận vào một event, nếu đây là sự kiện người dùng thực hiện một nước đi thì sẽ gọi đến hàm thực hiện nước đi của máy.

- Hàm human_turn(event): hàm xử lý tất cả các sự kiện từ người chơi liên quan đến bàn cờ: chọn, huỷ chọn, di chuyển...

- Hàm bot_turn(): là hàm thực hiện nước đi hợp lý của máy.

Class Bot: bao gồm các hàm hỗ trợ cho việc tính toán nước đi của máy

- Biến MAX_DEPTH: là giá trị độ sâu mà hàm sẽ thực hiện tìm kiếm.

- Mảng table[][]: là mảng chứa giá trị random sử dụng cho zobrish hashing.

- Map tranpos_table: bảng chuyển vị (tranposition table), là một map lưu giá trị tương ứng với trạng thái bàn cờ.

- Con trỏ hàm calculate_score: khi ở trạng thái đầu và giữa game thì calculate_score = calculate_score_nomal, còn khi ở trạng thái cuối game thì calculate_score = calculate_score_last_game.

- Map pv_move: là map lưu các nước đi tốt (pv-moves) cho một trạng thái.

- Hàm init_zobrish(): là hàm khởi tạo các giá trị random 64-bits cho mảng table 12x64, sử dụng để tạo Zobrish hash

- Hàm get_hash(board): là hàm tính giá trị hash cho trạng thái hiện tại của bàn cờ (chương trình không sử dụng hàm này)

- Hàm calculate_hash(move): hàm tính giá trị hash thay đổi nếu đi nước đi move

- Hàm calculate_score_normal(move): hàm tính sự thay đổi của giá trị heuristic nếu thực hiện nước đi move trong điều kiện đầu và giữa game.

- Hàm calcute_score_last_game(move): hàm tính sự thay đổi của giá trị hẻuistic nếu đi nước đi move trong điều kiện cuối game game.

- Hàm get_score(): tính điểm của trạng thái hiện tại (hiện không được sử dụng trong chương trình)

- Hàm iterative_deepening(): hàm thực hiện đào sâu tăng dần

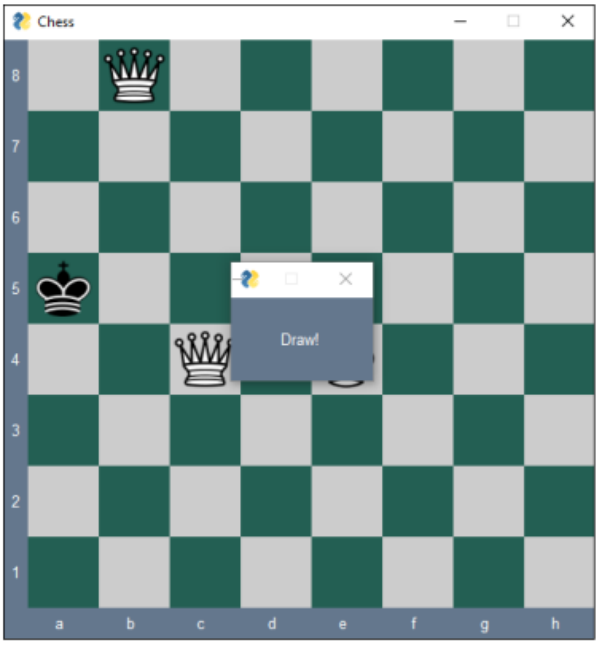
- Hàm minimax(depth, alpha, beta, isMaxPlayer): hàm triển khai thuật toán minimax và các thuật toán tối ưu

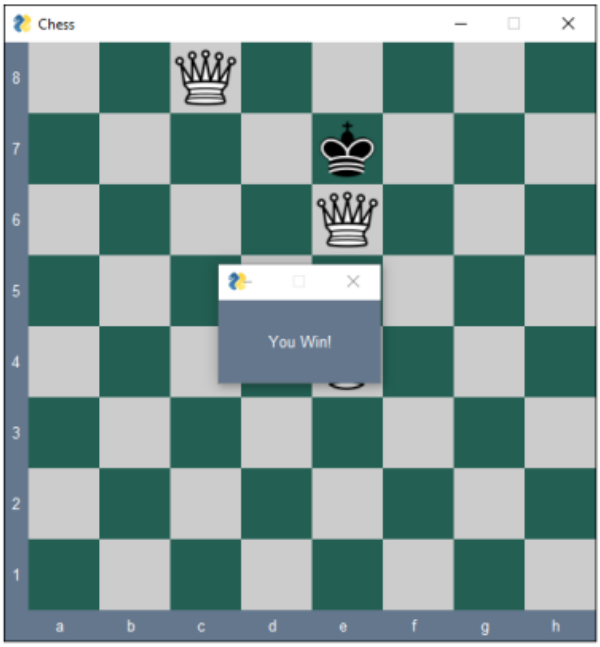
3.3. Giao diện chương trình

Giao diện khi mới khởi động chương trình:

Giao diện chọn quân cờ để phong khi quân tốt phong cấp:

Giao diện khi kết thúc ván cờ:





KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong quá trình thực hiện và hoàn thành đồ án môn học, chúng em đã hoàn thiện đầy đủ các yêu cầu đề ra trong bài toán. Đồ án môn học giúp chúng em hiểu rõ về thuật toán minimax, các thuật toán tối ưu, biết thêm về ngôn ngữ python, biết các thư viện và cách sử dụng…

Ngoài những ưu điểm trên thì trong thời gian phát triển, do thời gian và kiến thức có hạn nên trong chương trình có thể còn tồn tại một vài lỗi cơ bản mà chúng em chưa kiểm soát được, bố cục của chương trình chưa thực sự rõ ràng, chương trình chưa thực sự tối ưu.

Trong tương lai, dựa vào những kinh nghiệm đã có được qua đồ án môn học, chúng em chắc chắn sẽ khắc phục được những nhược điểm nêu trên. Và nếu có điều kiện cho phép về thời gian, nhân lực, nhóm có thể phát chương trình thực sự tối ưu về cả đồ họa và thuật toán, có thể thành một sản phẩm có giá trị hơn.

TÀI LIỆU THAM KHẢO

[1] Slide bài giảng môn Trí tuệ nhân tạo

[2] python-chess: a chess library for Python

https://python-chess.readthedocs.io/en/latest/

[3] Introduction to PySimpleGUI

https://www.geeksforgeeks.org/introduction-to-pysimplegui/

[4] A step-by-step guide to building a simple chess AI

https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/

[5] Zobrish hashing

https://en.wikipedia.org/wiki/Zobrist_hashing

[6] Transposition Table

https://en.wikipedia.org/wiki/Transposition_table

[7] Principal Variation Search

https://www.chessprogramming.org/Principal_Variation_Search

[8] Stackoverflow