**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**ĐẠI HỌC KINH TẾ QUỐC DÂN**

**Trường Công nghệ**



A logo with a map and a red and blue circle

Description automatically generated

**Đề tài**

**Phát triển phần mềm cờ vua**

Họ và tên: Đinh Lương Đức Anh

MSV: 11210323

Lớp: CNTT1173 (123)\_01

Giảng viên hướng dẫn: TS. Phạm Minh Hoàn

Hà Nội, năm 2023

**LỜI CẢM ƠN**

Để có cơ hội nghiên cứu và xây dựng đề tài “*Phát triển phần mềm cờ vua*”, em xin chân thành cảm ơn Ban lãnh đạo nhà trường Đại học Kinh tế Quốc dân đã tạo điều kiện cho em được học tập tại trường Đại học Kinh tế Quốc dân**.**

Em có thể thực hiện được tốt việc nghiên cứu đề tài này, chắc chắn không thể không kể đến sự truyền đạt kiến thức và giúp đỡ của các thầy cô giảng viên từ Viện Công nghệ thông tin và Kinh tế số trường Đại học Kinh tế Quốc dân. Trước hết, em xin được bày tỏ lòng biết ơn vì sự quan tâm, chỉ dạy và giúp đỡ của các thầy, các cô thời gian qua.

Đặc biệt, em xin gửi lời cảm ơn sâu sắc nhất đến người đã trực tiếp hướng dẫn em hoàn thành đề tài, TS.Phạm Minh Hoàn. Thầy đã luôn sát sao quá trình tiến hành nghiên cứu của chúng em từ những ngày đầu, giúp chúng em hoàn thiện tốt nhất việc xây dựng nên thành quả này.

Em xin chân thành cảm ơn!

MỤC LỤC

[DANH MỤC HÌNH ẢNH 2](#_Toc197927927)

[1.1. MỞ ĐẦU 4](#_Toc197927928)

[1.1.1. Giới thiệu chung 4](#_Toc197927929)

[1.1.2. Các nghiên cứu trong và ngoài nước 8](#_Toc197927930)

[1.1.3. Lý do chọn lựa đề tài 9](#_Toc197927931)

[1.1.4. Phạm vi và đối tượng nghiên cứu 9](#_Toc197927932)

[1.2. TỔNG QUAN VỀ HỌC MÁY (MACHINE LEARNING) 10](#_Toc197927933)

[1.2.1. Giới thiệu về học máy 10](#_Toc197927934)

[1.2.2. Phân loại học máy 10](#_Toc197927935)

[1.2.3. Cấu trúc học máy (Admin1, 2023) 11](#_Toc197927936)

[CHƯƠNG 2: Mô tả và phân tích bài toán 12](#_Toc197927937)

[2.1. TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH PYTHON 12](#_Toc197927938)

[2.1.1. Khái niệm 12](#_Toc197927939)

[2.1.2. Đặc điểm của ngôn ngữ Python 12](#_Toc197927940)

[2.1.3. Một số hàm tích hợp sẵn 12](#_Toc197927941)

[2.1.4. Thư viện Tkinter 15](#_Toc197927942)

[2.2. Sơ đồ và các giả mã cho các thuật toán 19](#_Toc197927943)

[2.2.1. Sơ đồ thuật toán cho thuật toán Minimax 19](#_Toc197927944)

[2.2.2. Giả mã cho thuật toán Minimax 19](#_Toc197927945)

[CHƯƠNG 3: Phương pháp/Kỹ thuật 31](#_Toc197927946)

[3.1. Phương pháp 31](#_Toc197927947)

[3.1.1. Mô hình giải quyết bài toán 31](#_Toc197927948)

[3.1.2. Thiết kế bàn cờ vua 50](#_Toc197927949)

[CHƯƠNG 5: Tổng kết 60](#_Toc197927950)

[KẾT QUẢ ĐẠT ĐƯỢC 60](#_Toc197927951)

[ĐIỂM CẦN CẢI THIỆN 60](#_Toc197927952)

[HƯỚNG PHÁT TRIỂN 60](#_Toc197927953)

[TÀI LIỆU THAM KHẢO 62](#_Toc197927954)

# DANH MỤC HÌNH ẢNH

[Hình 1. Minh họa nước đi của quân Tốt 4](#_Toc197927955)

[Hình 2. Minh họa nước đi của quân Xe 5](#_Toc197927956)

[Hình 3. Minh họa nước đi của quân Mã 5](#_Toc197927957)

[Hình 4. Minh họa nước đi của quân Tượng 6](#_Toc197927958)

[Hình 5. Minh họa nước đi của quân Hậu 6](#_Toc197927959)

[Hình 6. Minh họa nước đi của quân Vua 7](#_Toc197927960)

[Hình 7. Nhập thành gần 0-0 8](#_Toc197927961)

[Hình 8. Nhập thành xa 0-0-0 8](#_Toc197927962)

[Hình 9. Minh họa nước đi ăn Tốt qua đường 8](#_Toc197927963)

[Hình 2.1‑1. Giao diện ví dụ của Tkinter 18](#_Toc197927964)

[Hình 11. Sơ đồ thuật toán Minimax 19](#_Toc197927965)

[Hình 12. Sơ đồ thuật toán phương pháp cắt tỉa Alpha-Beta 22](#_Toc197927966)

[Hình 13. Heatmap cho thuật toán Piece-Square Table cho Mã Trắng 25](#_Toc197927967)

[Hình 14. Sơ đồ mô tả thuật toán của phần mềm cờ vua (chức năng evaluate\_board). 31](#_Toc197927968)

[Hình 15. Sơ đồ evaluate\_board - lặp qua ô và tính điểm từng quân cờ 33](#_Toc197927969)

[Hình 16. Sơ đồ evaluate\_board - hoàn tất và trả về điểm số 34](#_Toc197927970)

[Hình 17. Sơ đồ hàm make\_move 36](#_Toc197927971)

[Hình 18. Sơ đồ hàm alphabeta 38](#_Toc197927972)

[Hình 19. Hàm is\_king\_in\_check 39](#_Toc197927973)

[Hình 20. Sơ đồ hàm generate\_pseudo\_legal\_moves 40](#_Toc197927974)

[Hình 21. Hàm get\_legal\_moves 42](#_Toc197927975)

[Hình 22. Sơ đồ hàm is\_square\_attacked (Luồng chính) 44](#_Toc197927976)

[Hình 23. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công Tốt 45](#_Toc197927977)

[Hình 24. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công Mã 46](#_Toc197927978)

[Hình 25. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công quân trượt (Xe, Tượng, Hậu) 48](#_Toc197927979)

[Hình 26. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công quân Vua 49](#_Toc197927980)

[Hình 27. Giao diện người dùng. 50](#_Toc197927981)

[Hình 28. Hàm draw\_board, 50](#_Toc197927982)

[Hình 29. Sơ đồ hàm nước đi cho quân Mã. 52](#_Toc197927983)

[Hình 30. Sơ đồ hàm tạo nước đi cho quân Xe. 53](#_Toc197927984)

[Hình 31. Sơ đồ hàm tạo nước đi cho quân Tượng 54](#_Toc197927985)

[Hình 32. Sơ đồ hàm tạo nước đi cho quân Hậu. 55](#_Toc197927986)

[Hình 33. Sơ đồ hàm tạo nước đi cho quân Vua. 56](#_Toc197927987)

[Hình 34. Sơ đồ hàm tạo nước đi cho quân Tốt. 57](#_Toc197927988)

**CHƯƠNG 1****: Tổng quan tình hình nghiên cứu thuộc lĩnh vực đề tài**

# MỞ ĐẦU

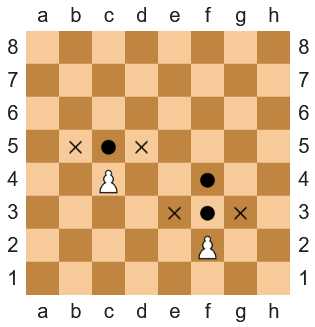
## Giới thiệu chung

Cờ vua là một trò chơi board game trí tuệ dành cho 2 người, không chứa yếu tố may mắn và dựa thuần túy vào chiến thuật và chiến lược, có nguồn gốc từ Nam Âu vào nửa sau của thế kỉ XV, mặc dù có nhiều tương đồng với trò chơi cổ chaturanga có nguồn gốc ở Ấn Độ. (Wikipedia.org, n.d.)

Bàn cờ vua gồm 8 cột (files) và 8 hàng (ranks), gộp lại tạo thành 64 ô trên bàn cờ, 2 người chơi được chia thành 2 phe Trắng (White) và Đen (Black), mỗi phe điều khiển 16 “quân” theo lượt, bên trắng luôn được đi trước, trong đó có 8 quân Tốt/Chốt (Pawn), 2 quân Xe/Xa (Rook), 2 quân Mã (Knight), 2 quân Tượng (Bishop), 1 quân Hậu (Queen) và 1 quân Vua (King). Các quân có thể bị “ăn” (captured) nếu có một quân địch đi vào ô vuông quân đó đang chiếm giữ.

Các quân có các nước đi khác nhau:

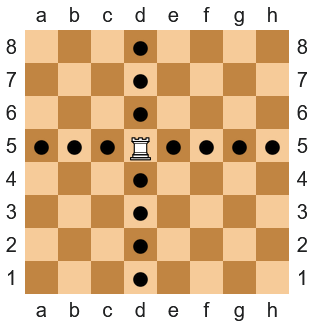
* Quân Tốt:



Hình 1. Minh họa nước đi của quân Tốt

Có thể di chuyển thẳng về phía trước 1 hoặc 2 ô nếu ở vị trí xuất phát (hàng 2 cho bên Trắng và hàng 7 cho bên Đen), không thể nhảy qua đầu một quân khác và khi ăn quân đối phương sẽ ăn theo đường chéo (biểu thị bằng dấu X trên hình).

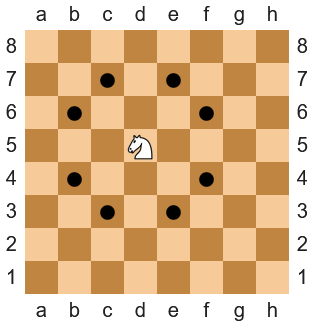
* Quân Xe:



Hình 2. Minh họa nước đi của quân Xe

Có thể di chuyển theo các đường thẳng dọc theo các cột hoặc hàng tới ô còn trống mà không có quân nào cản trên đường đi hay tới ô bị quân đối phương chiếm giữ (ăn quân) nhưng không thể vượt qua quân đang đứng ở ô đó.

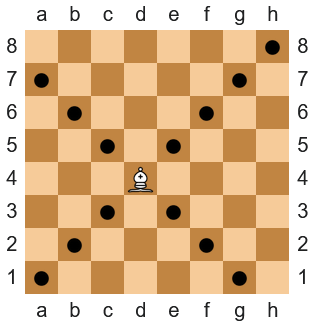
* Quân Mã:



Hình 3. Minh họa nước đi của quân Mã

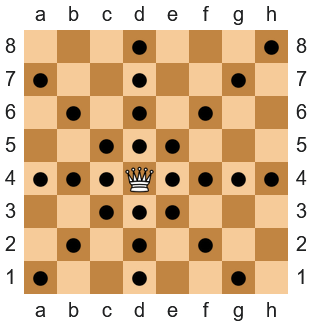
Có thể di chuyển tới ô còn trống hay ô bị quân đối phương chiếm giữ (ăn quân) theo dạng hình chữ L (hình chữ nhật 3×2 hay 2×3). Quân mã không thể bị cản trừ khi nước đi đó khiến quân Vua bị chiếu.

* Quân Tượng:



Hình 4. Minh họa nước đi của quân Tượng

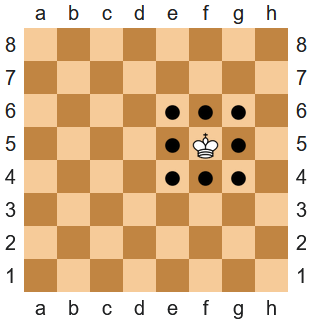
Có thể di chuyển theo đường chéo tới ô có cùng màu với nguyên lý tương tự như xe tới ô còn trống hay ô bị quân đối phương chiếm giữ (ăn quân).

* Quân Hậu:
* 

Hình 5. Minh họa nước đi của quân Hậu

Có nước đi là tổ hợp đơn giản của chuyển động của xe và tượng. Trong một nước đi nó có thể di chuyển theo đường chéo hoặc đường thẳng dọc theo cột hay hàng, với nguyên lý đi và ăn quân giống như tượng và xe.

* Quân Vua:



Hình 6. Minh họa nước đi của quân Vua

Là quân quan trọng nhất, nếu mất vua thì người chơi thua cuộc. Mỗi lần đi nó có thể ăn quân hoặc di chuyển sang các ô bao quanh ô mà nó hiện tại đang chiếm giữ, nhưng không thể tới ô mà quân của mình đang chiếm giữ hay các ô bị quân đối phương kiểm soát.

Mục tiêu của người chơi là cố gắng “chiếu hết” (checkmate) vua đối phương, có nghĩa là chiếu (check) mà không có cách nào để thoát ra. Khi bị một người chơi bị chiếu hết, trò chơi sẽ kết thúc và người chơi này sẽ bị tính là thua cuộc

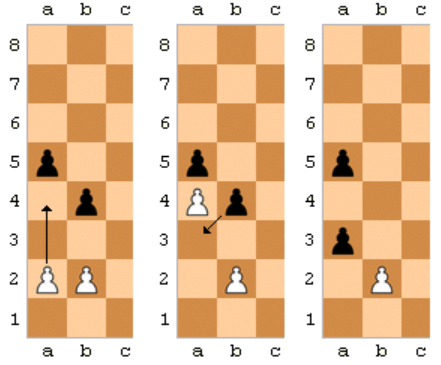
Một số nước đi đặc biệt của cờ vua bao gồm:

* Phong cấp (Promotion): Một tốt được phong cấp khi nó đã đi đến hàng cuối cùng của bàn cờ đối diện với nó. Khi đó, nó sẽ trở thành một trong các quân Hậu, Tượng, Mã hoặc Xe cùng màu theo ý muốn của người chơi. (FIDE, 2022) Tốt được phong cấp sẽ biến thành quân cờ mới ngay trước nước đi tiếp theo của đối phương. Nhưng trong đề án này tốt chỉ có thể nâng thành quân Hậu để đơn giản hóa bài toán.
* Nhập thành (Castling): Vua và một trong hai quân Xe tham gia vào việc di chuyển. Khi nhập thành Vua di chuyển qua 2 ô về phía quân Xe tham gia nhập thành, và sau đó di chuyển quân Xe tới ô mà quân Vua vừa di chuyển qua sao cho nó nằm ngay bên cạnh quân Vua.

Việc nhập thành chỉ được phép nếu cả quân Vua và quân Xe chưa từng di chuyển trước đó, các ô giữa quân Vua và quân Xe không có quân nào nằm giữa; các ô mà Vua sẽ di chuyển qua không nằm dưới sự kiểm soát (ô hay đường nằm trong tầm chiếu) của quân đối phương, cũng như việc nhập thành không làm được khi bị chiếu. (FIDE, 2022)

|  |  |
| --- | --- |
| Hình 7. Nhập thành gần 0-0 | Hình 8. Nhập thành xa 0-0-0 |

* Bắt Tốt qua đường (En passant): xảy ra khi quân Tốt bắt quân Tốt vừa di chuyển hai ô trong lượt đầu tiên theo hướng chéo lên. (FIDE, 2022) Quân Tốt sau khi thực hiện en passant sẽ ở vị trí mà nếu như con Tốt bị bắt ban đầu chỉ di chuyển một ô.



Hình 9. Minh họa nước đi ăn Tốt qua đường

## Các nghiên cứu trong và ngoài nước

Lĩnh vực phát triển phần mềm cờ vua và trí tuệ nhân tạo trong cờ vua đã có lịch sử phát triển lâu dài và là một trong những trường hợp nổi tiếng nhất về việc trí tuệ nhân tạo đánh bại con người. Năm 1997 khi DeepBlue của IBM đánh bại nhà vô địch thế giới Garry Kasparov và vào năm 2005, trận thua 2-4 của nhà vô địch thế giới lúc bấy giờ, Vladimir Borisovich Kramnik với Deep Fritz đã đánh dấu trận đấu lớn cuối cùng giữa máy và người cũng như là thời điểm máy tính chính thức vượt qua con người trong môn thể thao trí tuệ này.

Hiện nay, các phần mềm cờ vua hiện đại như Stockfish, LCZero và AlphaZero cũng đều đạt đạt đến cấp độ siêu đại kiện tướng với điểm Elo vượt trên 3500 vào năm 2023 trong khi điểm Elo cao nhất con người là 2882, đạt được bởi Đại kiện tướng Magnus Carlsen (Lewis, 2020).

Các thuật toán truyền thống trong lĩnh vực này chủ yếu dựa trên cây quyết định Minimax và cắt tỉa Alpha-Beta (Alpha-Beta prunning) (Donald E. Knuth, 1975). Gần đây, cuộc cách mạng học sâu đã mang đến những đột phá với AlphaZero của Google DeepMind, sử dụng mạng nơ-ron kết hợp với tìm kiếm cây Monte Carlo để đạt hiệu suất vượt trội mà không cần kiến thức chuyên gia (David Silver, 2018).

Tại Việt Nam, mặc dù chưa có nhiều nghiên cứu chuyên sâu về AI trong cờ vua nhưng đã có những dự án học thuật và phát triển phần mềm cờ vua trong các đồ án chuyên ngành hoặc nghiên cứu học thuật (PGS. TS. Phạm Văn Cường, 2023).

## Lý do chọn lựa đề tài

Cờ vua là một trong những lĩnh vực được nghiên cứu lâu đời nhất trong lịch sử của trí tuệ nhân tạo và với đó là tài liệu nghiên cứu đầy đủ và phong phú, cùng với nhiều ví dụ trực quan và sinh động. Cờ vua cũng là một môi trường lý tưởng để nghiên cứu và áp dụng các thuật toán trí tuệ nhân tạo vì quy luật rõ ràng nhưng không gian trạng thái cực kỳ phức tạp với khoảng 10^120 bước đi hợp lệ, góp phần tạo ra một thách thức lớn và thú vị cho việc phát triển một thuật toán hiệu quả trong thời gian cho sẵn. Đây là một cơ hội để phát triển và tìm hiểu sâu hơn về các thuật toán tìm kiếm và đánh giá suy nghiệm (heuristic) cũng như là áp dụng kiến thức lý thuyết về trí tuệ nhân tạo vào một ứng dụng thực tế

## Phạm vi và đối tượng nghiên cứu

Mục đích chính của nghiên cứu này là tìm hiểu và phân tích cơ sở lý thuyết và cách hoạt động của thuật toán Minimax và cắt tỉa Alpha-Beta trong lĩnh vực học máy. Nghiên cứu sẽ tập trung vào việc hiểu cách Minimax thực hiện việc chọn nước đi tối ưu nhất sau khi phân tích và đánh giá bàn cờ bằng cách tính điểm theo nguyên tắc Min-Max và dự đoán nước đi trước bằng cách mô phỏng tất cả các nước đi hợp lệ trong nhiều lượt tiếp theo tùy theo giá trị “độ sâu” (depth) của thuật toán cùng với một số hàm suy nghiệm cơ bản theo phương pháp Piece-Square Table (PST)

Nghiên cứu sẽ tiến hành cài đặt và triển khai giải thuật Minimax và cắt tỉa Alpha-Beta trong môi trường máy tính qua phần mềm đánh cờ vua chạy bằng ngôn ngữ Python. Các thử nghiệm sẽ được tiến hành qua người đánh với máy hoặc giả lập máy đánh với máy.

Một phần quan trọng của nghiên cứu này là xây dựng một phần mềm cờ vua có một đối thủ dựa trên học máy, trong đó giải thuật Minimax, cắt tỉa Alpha-Beta và thuật toán suy nghiệm PST sẽ được tích hợp. Phần mềm này có khả năng thực hiện các nhiệm vụ lưu trữ và dự đoán nước đi dựa trên dữ liệu đầu vào, và sẽ cung cấp giao diện người dùng dễ sử dụng.

Mục tiêu cuối cùng của nghiên cứu là đánh giá khả năng áp dụng giải thuật Minimax, cắt tỉa Alpha-Beta, giải thuật suy nghiệm PST của đối thủ máy trong phần mềm cờ vua. Nghiên cứu sẽ tập trung vào việc tìm hiểu cách Minimax, cắt tỉa Alpha-Beta và giải thuật suy nghiệm PST có thể giúp trong việc ra quyết định bằng cách phân tích dữ liệu bàn cờ, tính điểm và dự đoán nước đi bằng cách mô phỏng Minimax.

# TỔNG QUAN VỀ HỌC MÁY (MACHINE LEARNING)

## Giới thiệu về học máy

Học máy (Machine Learning - ML) là một phân nhánh của trí tuệ nhân tạo (Artificial Intelligence - AI) trong đó các hệ thống máy tính được thiết kế để tự học hỏi từ dữ liệu và tự cải thiện hiệu suất theo thời gian. Mục đích cốt lõi của học máy là xây dựng các mô hình và thuật toán giúp máy tính tự động thực hiện các nhiệm vụ như dự đoán, phân loại và khai thác thông tin từ dữ liệu mà không cần được lập trình chi tiết. (Trang, 2021)

Sức mạnh của các thuật toán học máy nằm ở khả năng phát hiện mối tương quan, xu hướng và cấu trúc trong dữ liệu. Chúng có thể học từ dữ liệu đầu vào, trích xuất đặc trưng, nắm bắt các quy luật ẩn và dự đoán hoặc phân loại các kết quả mới. Các ví dụ điển hình về thuật toán học máy bao gồm cây quyết định, máy vector hỗ trợ và mạng nơ-ron nhân tạo. (Bình, 2023)

## Phân loại học máy

Học máy được chia thành hai loại chính: ***học máy có giám sát*** và ***học máy không giám sát***.

**Học có máy giám sát (Supervised Learning)**: Trong phương pháp này, dữ liệu huấn luyện được cung cấp kèm với nhãn (label) đã biết trước. Mô hình học máy được đào tạo để tìm ra mối liên hệ giữa các đặc trưng (feature) và nhãn, từ đó có khả năng dự đoán nhãn cho dữ liệu mới. Các ứng dụng của học có giám sát bao gồm phân loại (classification) và hồi quy (regression).

**Học máy không giám sát (Unsupervised Learning)**: Trong cách tiếp cận này, dữ liệu huấn luyện không đi kèm nhãn. Mô hình học máy được sử dụng để khám phá cấu trúc tiềm ẩn, mô hình hóa dữ liệu và tìm ra các nhóm hoặc mẫu trong tập dữ liệu. Các ứng dụng tiêu biểu của học không giám sát bao gồm phân cụm (clustering) và giảm chiều dữ liệu (dimensionality reduction).

Ngoài ra, học máy còn liên quan đến các phương pháp khác như học tăng cường (reinforcement learning) và học sâu (deep learning). Học máy được ứng dụng rộng rãi trong nhiều lĩnh vực như xử lý ngôn ngữ tự nhiên, thị giác máy tính, nhận dạng giọng nói và nhiều ứng dụng khác.

## Cấu trúc học máy (Admin1, 2023)

Dữ liệu (Data): Dữ liệu đóng vai trò then chốt trong học máy. Nó bao gồm tập dữ liệu huấn luyện, tập dữ liệu kiểm thử và tập dữ liệu đánh giá. Dữ liệu có thể là các đặc trưng (features) mô tả thuộc tính của đối tượng hoặc nhãn (label) gắn liền với đối tượng đó.

Tiền xử lý dữ liệu (Data Preprocessing): Trước khi huấn luyện mô hình, dữ liệu thường cần được xử lý sơ bộ để làm sạch, chuẩn hóa và trích xuất đặc trưng. Quá trình này bao gồm các công đoạn như xử lý dữ liệu thiếu, mã hóa biến phân loại, chuẩn hóa dữ liệu số, và loại bỏ nhiễu.

Mô hình học máy (Machine Learning Models): Mô hình học máy là một thuật toán hoặc tập hợp thuật toán được dùng để học từ dữ liệu huấn luyện và thực hiện dự đoán hoặc phân loại cho dữ liệu mới. Các mô hình học máy phổ biến có thể kể đến như cây quyết định, máy vector hỗ trợ, mạng nơ-ron nhân tạo, và nhiều mô hình khác.

Huấn luyện (Training): Quá trình huấn luyện mô hình liên quan đến việc đưa dữ liệu huấn luyện vào mô hình và điều chỉnh các tham số để tối ưu hóa hiệu suất. Quá trình này thường dựa vào một hàm mất mát (loss function) để đánh giá sai số giữa dự đoán và giá trị thực tế, kết hợp với thuật toán tối ưu hóa để điều chỉnh tham số.

Đánh giá (Evaluation): Sau khi mô hình đã được huấn luyện, cần được đánh giá thông qua tập dữ liệu kiểm thử hoặc tập dữ liệu đánh giá. Việc đánh giá giúp đo lường hiệu suất và độ chính xác của mô hình, đồng thời xác định liệu mô hình có hoạt động tốt trên dữ liệu mới hay không.

Điều chỉnh (Tuning): Điều chỉnh mô hình là quá trình tìm kiếm bộ tham số tối ưu nhằm nâng cao hiệu suất. Quá trình này thường áp dụng kỹ thuật kiểm chứng chéo (cross-validation) và tìm kiếm lưới (grid search) để xác định tổ hợp tham số hiệu quả nhất.

Triển khai (Deployment): Sau khi mô hình đã được huấn luyện và tinh chỉnh, có thể triển khai trong môi trường thực tế để thực hiện dự đoán hoặc phân loại trên dữ liệu mới. Việc triển khai có thể là tích hợp mô hình vào ứng dụng, hệ thống, hoặc dịch vụ thực tế.

# CHƯƠNG 2: Mô tả và phân tích bài toán

# TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH PYTHON

## Khái niệm

Python đại diện cho một ngôn ngữ lập trình bậc cao, đa năng. Nó được xây dựng với những đặc điểm tích cực như dễ đọc, dễ học và dễ nhớ. Cấu trúc rõ ràng của Python làm cho nó trở nên thuận tiện cho người mới bắt đầu trong lập trình, giảm thiểu số lần gõ phím cần thiết.

Python thực hiện kiểu động và sử dụng cơ chế cấp phát bộ nhớ tự động, tương tự như các ngôn ngữ như Perl, Ruby, Scheme, Smalltalk và Tcl. Dự án Python được phát triển trong bối cảnh mã nguồn mở, và Python Software Foundation, một tổ chức phi lợi nhuận, đảm bảo quản lý và phát triển ngôn ngữ này.

## Đặc điểm của ngôn ngữ Python

* Ngôn ngữ lập trình đơn giản, dễ học
* Miễn phí, mã nguồn mở
* Khả năng di động linh hoạt
* Khả năng mở rộng và có thể nhúng
* Ngôn ngữ thông dịch cấp cao
* Thư viện tiêu chuẩn lớn để giải quyết những tác vụ phổ biến
* Hướng đối tượng

## Một số hàm tích hợp sẵn

**print()**: Dùng để hiển thị thông tin ra màn hình.

print("Hello, World!") # Hello, World!

**len()**: Trả về độ dài của một chuỗi, danh sách, hoặc bất kỳ đối tượng nào có thể đếm được.

my\_list = [1, 2, 3, 4, 5]

length = len(my\_list)

print(length) # Kết quả: 5

**input()**: Sử dụng để nhận dữ liệu đầu vào từ người dùng qua bàn phím.

user\_input = input("Nhập tên của bạn: ")

print("Xin chào, " + user\_input + "!")

type(): Trả về kiểu dữ liệu của một đối tượng.

x = 5

print(type(x)) # <class 'int'>

**int()**, **float()**, **str()**, **list()**, **dict()**, **tuple()**: Dùng để ép kiểu đối tượng sang kiểu dữ liệu khác nhau.

num\_str = "42"

num\_int = int(num\_str)

print(num\_int) # 42

my\_float = float("3.14")

print(my\_float) # 3.14

range(): Tạo ra một dãy số nguyên liên tiếp.

my\_range = range(1, 6)

print(list(my\_range)) # [1, 2, 3, 4, 5]

**max()** và **min()**: Trả về giá trị lớn nhất và nhỏ nhất trong một chuỗi hoặc danh sách.

my\_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

max\_value = max(my\_list)

min\_value = min(my\_list)

print("Max:", max\_value) # 9

print("Min:", min\_value) # 1

sum(): Tính tổng các phần tử trong một chuỗi hoặc danh sách.

my\_list = [1, 2, 3, 4, 5]

total = sum(my\_list)

print(total) # 15

**abs()**: Trả về giá trị tuyệt đối của một số.

x = -5

absolute\_value = abs(x)

print(absolute\_value) # 5

**sorted()**: Sắp xếp một chuỗi hoặc danh sách.

my\_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

sorted\_list = sorted(my\_list)

print(sorted\_list) # [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

**format()**: Dùng để định dạng chuỗi với các giá trị được chèn vào vị trí cố định trong chuỗi.

name = "Alice"

age = 30

formatted\_string = "Xin chào, tôi là {} và tôi {} tuổi.".format(name, age)

print(formatted\_string) # "Xin chào, tôi là Alice và tôi 30 tuổi."

**zip()**: Ghép các phần tử từ nhiều danh sách thành các cặp (tuples).

list1 = [1, 2, 3]

list2 = ['a', 'b', 'c']

zipped = list(zip(list1, list2))

print(zipped) # [(1, 'a'), (2, 'b'), (3, 'c')]

**enumerate()**: Trả về cặp chỉ số và giá trị từ một chuỗi hoặc danh sách.

my\_list = ['a', 'b', 'c'] for index, value in enumerate(my\_list):

print("Index:", index, "Value:", value)

# Kết quả:

# Index: 0 Value: a

# Index: 1 Value: b

# Index: 2 Value: c

**map()**: Áp dụng một hàm lên từng phần tử của một danh sách.

def square(x):

return x \*\* 2

my\_list = [1, 2, 3, 4, 5]

squared\_list = list(map(square, my\_list))

print(squared\_list) # [1, 4, 9, 16, 25]

**filter()**: Lọc ra các phần tử của một danh sách thỏa mãn một điều kiện nào đó.

def is\_even(x):

return x % 2 == 0

my\_list = [1, 2, 3, 4, 5]

even\_numbers = list(filter(is\_even, my\_list))

print(even\_numbers) # [2, 4]

**any()** và **all()**: Kiểm tra xem có ít nhất một phần tử thỏa mãn điều kiện hoặc tất cả phần tử đều thỏa mãn điều kiện trong một chuỗi hoặc danh sách.

my\_list = [True, False, True]

result\_any = any(my\_list)

result\_all = all(my\_list)

print(result\_any) # True (vì có ít nhất một giá trị True)

print(result\_all) # False (vì không phải tất cả giá trị đều True)

**dir()**: Liệt kê tất cả các thuộc tính và phương thức của một đối tượng (ví dụ này phụ thuộc vào đối tượng cụ thể bạn muốn kiểm tra).

my\_list = [1, 2, 3]

# Sử dụng hàm dir() để liệt kê tất cả các thuộc tính và phương thức của đối tượng my\_list

attributes\_and\_methods = dir(my\_list)

# In danh sách các thuộc tính và phương thức

print(attributes\_and\_methods)

## Thư viện Tkinter

Tkinter là một thư viện GUI (Graphical User Interface) mặc định đi kèm với Python, giúp nhà phát triển phần mềm tạo giao diện người dùng cho ứng dụng và chương trình của mình. Tên "Tkinter" là viết tắt của "Tk Interface," và nó được xây dựng trên nền tảng Tk, một toolkit GUI đa nền tảng, giúp bạn tạo các ứng dụng giao diện sử dụng đồ họa và tương tác với người dùng một cách dễ dàng.

**Chức năng:**

**Tạo các thành phần giao diện người dùng (Widgets):** Tkinter cung cấp nhiều loại widget như nút (Button), nhãn (Label), ô văn bản (Entry), danh sách (Listbox), thanh cuộn (Scrollbar), cửa sổ (Window), hộp thoại thông báo (MessageBox), và nhiều widget khác. Bạn có thể sử dụng những widget này để xây dựng giao diện người dùng đa dạng.

**Quản lý bố cục (Layout Management):** Tkinter cung cấp các phương pháp quản lý bố cục để sắp xếp và điều chỉnh vị trí của các widget trong cửa sổ hoặc khung (frame). Ba phương pháp quản lý bố cục chính là pack(), grid(), và place().

**Sự kiện và xử lý sự kiện:** Tkinter cho phép bạn định nghĩa và xử lý sự kiện như nhấn nút, di chuyển chuột, và nhấn phím. Bạn có thể gắn các hàm xử lý sự kiện vào các widget để phản hồi khi người dùng tương tác với ứng dụng của bạn.

**Thanh cuộn (Scrollbar):** Tkinter cho phép tạo các thanh cuộn ngang và dọc để xử lý nội dung vượt quá kích thước của cửa sổ hoặc khung. Điều này làm cho việc xem và tương tác với dữ liệu lớn trở nên dễ dàng hơn.

**Cửa sổ độc lập (Toplevel Windows):** Bạn có thể tạo nhiều cửa sổ độc lập trong cùng một ứng dụng, mỗi cửa sổ có thể chứa một phần giao diện người dùng riêng biệt. Điều này cho phép bạn tạo các hộp thoại, cửa sổ con, và ứng dụng đa cửa sổ.

**Tùy chỉnh giao diện người dùng:** Tkinter cho phép bạn tùy chỉnh giao diện người dùng bằng cách thay đổi màu sắc, kích thước, kiểu chữ, và hình dạng của các widget. Bạn có thể tạo các widget tùy chỉnh riêng của mình để phù hợp với thiết kế của bạn.

**Hỗ trợ cho đồ họa:** Tkinter có thể được sử dụng để vẽ đồ họa cơ bản, với khả năng vẽ các hình cơ bản như đường thẳng, hình chữ nhật, và hình tròn.

**Hỗ trợ văn bản và hình ảnh**: Bạn có thể hiển thị văn bản định dạng và hình ảnh trong các widget như nhãn và ô văn bản.

**Hộp thoại thông báo (MessageBox)**: Tkinter cung cấp hộp thoại thông báo tiêu chuẩn để hiển thị thông báo, cảnh báo, và yêu cầu xác nhận từ người dùng.

**Các lớp và phương thức:**

**Lớp Tk:** cửa sổ gốc của ứng dụng Tkinter. Bạn tạo một đối tượng Tk để tạo một cửa sổ ứng dụng.

**Phương thức Tk():** được sử dụng để tạo một đối tượng cửa sổ.

**Lớp Frame:** là một khung chứa widget, giúp bạn tổ chức các thành phần trên giao diện người dùng.

**Phương thức Frame()**: được sử dụng để tạo một đối tượng Frame.

**Lớp Label:** cho phép bạn hiển thị văn bản hoặc hình ảnh tĩnh trên giao diện người dùng.

**Phương thức Label(parent, text="")**: tạo một đối tượng nhãn với văn bản hoặc hình ảnh đã chỉ định.

**Lớp Button:** tạo ra nút có khả năng thực hiện một hành động khi được nhấn.

**Phương thức Button(parent, text="", command=callback):** tạo một đối tượng nút với văn bản và hàm gọi lại khi nút được nhấn.

**Lớp Entry:** là một ô văn bản cho phép người dùng nhập dữ liệu văn bản.

**Phương thức Entry(parent)** tạo một đối tượng ô văn bản.

**Lớp Listbox:** tạo một danh sách dọc để hiển thị danh sách các mục.

**Phương thức Listbox(parent)** tạo một đối tượng danh sách.

**Lớp Scrollbar:** tạo thanh cuộn dọc hoặc ngang để xem nội dung vượt ra ngoài khu vực hiển thị.

**Phương thức Scrollbar(parent, orient=tk.VERTICAL):** tạo một đối tượng thanh cuộn dọc.

**Phương thức pack()**: được sử dụng để đặt widget vào bố cục giao diện người dùng dựa trên phương pháp đóng gói (packing). Widget sẽ tự động thích ứng với kích thước và vị trí trong khung cha.

**Phương thức grid():** cho phép bạn đặt widget trong lưới (grid) để tự do điều chỉnh vị trí và kích thước của widget trong khung cha.

**Phương thức bind(event, handler)**: cho phép bạn gắn một hàm xử lý sự kiện vào một widget. Widget sẽ gọi hàm này khi có sự kiện xảy ra, chẳng hạn như khi người dùng nhấn nút.

* Ví dụ:

import tkinter as tk

def button\_click():

entry\_text = entry.get()

label.config(text=f"Hello, {entry\_text}!")

root = tk.Tk()

root.title("Ví dụ sử dụng Tkinter")

# Tạo một khung chứa

frame = tk.Frame(root)

frame.pack()

# Tạo một nhãn

label = tk.Label(frame, text="Nhập tên:")

label.pack()

# Tạo một ô văn bản

entry = tk.Entry(frame)

entry.pack()

# Tạo một nút

button = tk.Button(frame, text="Xin chào", command=button\_click)

button.pack()

root.mainloop()

A screenshot of a computer

Description automatically generated

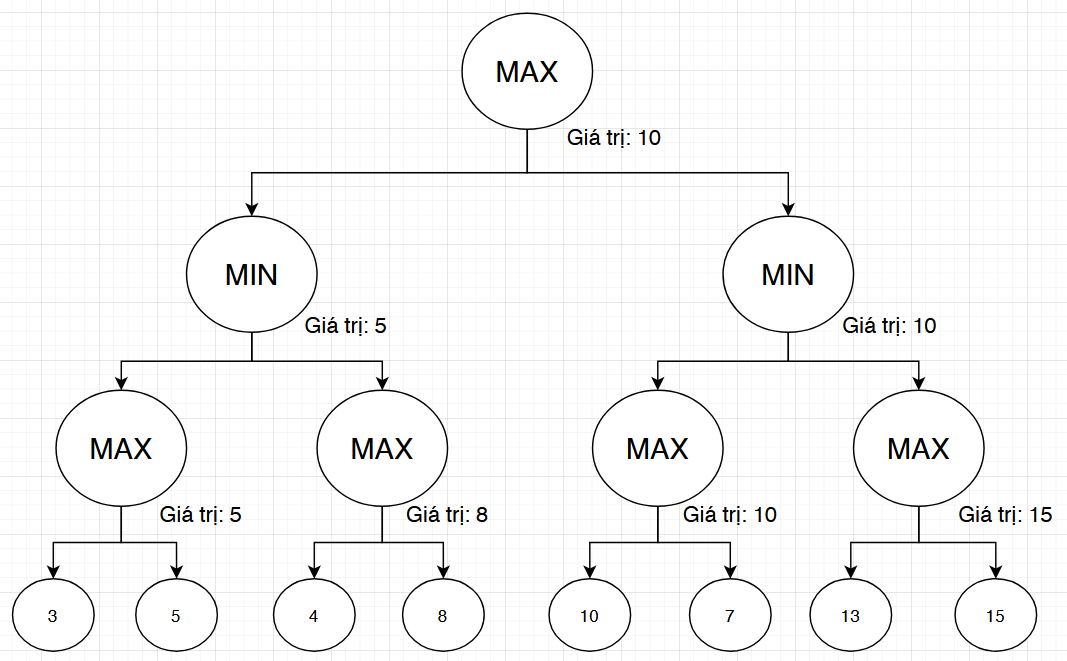
Hình 2.1‑1. Giao diện ví dụ của Tkinter

* Trong ví dụ này:
  + Chúng ta đã sử dụng lớp Tk để tạo cửa sổ ứng dụng.
  + Lớp Frame được sử dụng để tạo một khung chứa để chứa các widget khác.
  + Lớp Label được sử dụng để hiển thị một nhãn văn bản.
  + Lớp Entry được sử dụng để tạo một ô văn bản để nhập dữ liệu.
  + Lớp Button được sử dụng để tạo một nút có thể nhấn, và phương thức command được sử dụng để gắn hàm xử lý sự kiện vào nút.
  + Các phương thức pack() được sử dụng để đặt các widget vào bố cục của khung chứa.

Khi bạn chạy ví dụ này, bạn sẽ thấy một cửa sổ Tkinter với một ô văn bản và một nút "Xin chào". Khi bạn nhập tên vào ô văn bản và nhấn nút "Xin chào", một nhãn mới sẽ hiển thị chào mừng bạn theo tên bạn đã nhập.Bottom of Form.

## Sơ đồ và các giả mã cho các thuật toán

## Sơ đồ thuật toán cho thuật toán Minimax



Hình 11. Sơ đồ thuật toán Minimax

## Giả mã cho thuật toán Minimax

Thuật toán Minimax là một kỹ thuật đưa ra quyết định trong lý thuyết trò chơi, đặc biệt là cho các trò chơi hai người có thông tin đầy đủ và tổng bằng không (zero-sum games), nơi lợi ích của người này đồng nghĩa với tổn thất của người kia. Minimax được áp dụng rộng rãi trong trí tuệ nhân tạo cho các trò chơi như cờ vua, cờ tướng, cờ caro hay cờ carô.

**Nguyên lý cơ bản của thuật toán Minimax:**

Minimax ở cốt lõi dựa trên nguyên lý cơ bản là ý tưởng giả định đối thủ sẽ luôn chơi tối ưu, dẫn đến chiến lược “minimax” – tối thiểu hóa tổn thất tối đa có thể xảy ra

Minimax bắt đầu bằng việc xây dựng cây trò chơi, trong đó, mỗi nút đại diện cho một trạng thái trò chơi .

* Ở nút MAX: Người chơi tối đa hóa số điểm của mình.
* Ở nút MIN: Đối thủ cố gắng tối thiểu hóa điểm số của MAX.

Minimax xuất phát từ lý thuyết trò chơi của John Von Neumann. Định lý Minimax (1928) chứng minh rằng trong mọi trò chơi hai người tổng bằng không, có một "giá trị" V của trò chơi và một chiến lược tối ưu cho mỗi người chơi sao cho:

* Người chơi MAX có thể đảm bảo thắng ít nhất V điểm.
* Người chơi MIN có thể đảm bảo thua không quá V điểm.

Bắt đầu

function minimax(current\_state, search\_depth, is\_maximizing\_player):

if search\_depth == 0 or is\_terminal(current\_state):

return evaluate\_heuristic(current\_state)

if is\_maximizing\_player:

max\_eval = -infinity

for each next\_state achievable from current\_state:

eval = minimax(next\_state, search\_depth - 1, false)

max\_eval = max(max\_eval, eval)

return max\_eval

else:

min\_eval = +infinity

for each next\_state achievable from current\_state:

eval = minimax(next\_state, search\_depth - 1, true)

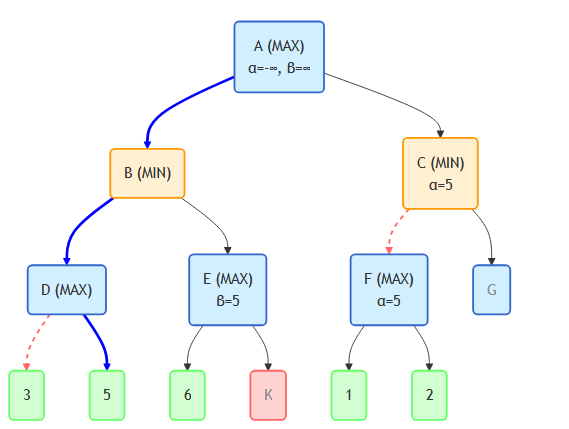
min\_eval = min(min\_eval, eval)

return min\_eval

Kết thúc

**Các biến có trong giả mã:**

* *current\_state*: Đây là trạng thái (thế cờ, tình huống) cụ thể mà hàm *minimax* đang xem xét tại thời điểm gọi hàm đó. Nó là điểm xuất phát để phân tích các nước đi tiếp theo.
* *search\_depth*: Xác định số lượt đi (hoặc số tầng trong cây trò chơi) mà thuật toán được phép nhìn về phía trước tính từ *current\_state*. Đây là một cách để giới hạn tính toán, tránh việc phải duyệt toàn bộ cây trò chơi.
* *is\_maximizing\_player*: Cho biết lượt đi tại *current\_state* này thuộc về người chơi nào.
* *max\_eval*: Biến này lưu trữ điểm số cao nhất tìm thấy được trong số tất cả các *next\_state* đã được xem xét từ *current\_state* (chỉ áp dụng cho người chơi Tối đa hóa).
* *min\_eval*: Biến này lưu trữ điểm số thấp nhất tìm thấy được trong số tất cả các *next\_state* đã được xem xét từ *current\_state* (chỉ áp dụng cho người chơi Tối thiểu hóa).
* *next\_state*: Đại diện cho một trạng thái trò chơi có thể đạt được từ *current\_state* sau đúng một nước đi hợp lệ.
* *eval*: Lưu trữ tạm thời giá trị (điểm số) được trả về từ một lệnh gọi đệ quy *minimax*(*next\_state*, ...). Đại diện cho kết quả dự đoán nếu lựa chọn đi đến next\_state đó.
  + 1. ***Sơ đồ thuật toán cho phương pháp cắt tỉa Alpha - Beta***



Hình 12. Sơ đồ thuật toán phương pháp cắt tỉa Alpha-Beta

Trong đó:

* Nút *maxNode* (xanh dương nhạt): Nút đại diện cho lượt đi của người chơi MAX. Mục tiêu là chọn giá trị lớn nhất từ các nút con.
* Nút *minNode* (Vàng nhạt): Nút đại diện cho lượt đi của người chơi MIN. Mục tiêu là chọn giá trị nhỏ nhất từ các nút con.
* Nút *leafNode* (Xanh lá nhạt): Nút lá, đại diện cho trạng thái kết thúc hoặc độ sâu tìm kiếm tối đa. Các giá trị số (3, 5, 6, 1, 2) là điểm số đánh giá tĩnh tại các lá này.
* *prunedNode* (Hồng nhạt): Nút và nút con đã bị cắt tỉa.
* Mũi tên:
  + Mũi tên đen: Thể hiện quá trình duyệt cây.
  + Mũi tên đỏ đứt nét (*prunedEdge*): Chỉ đến nhánh đã bị cắt tỉa.
  + Mũi tên xanh dương đậm (*finalPath*): Đánh dấu đường đi tối ưu dẫn đến kết quả cuối cùng.
    1. ***Giả mã cho kỹ thuật cắt tỉa Alpha - Beta***

Đây là một kỹ thuật tối ưu hóa cho thuật toán tìm kiếm Minimax. Mục tiêu của kỹ thuật này là giảm đáng kể số lượng nút cần phải duyệt và đánh giá trong cây tìm kiếm Minimax mà không làm ảnh hưởng đến kết quả cuối cùng (nước đi tối ưu).

**Nguyên lý cơ bản của kỹ thuật cắt tỉa Alpha – Beta:**

Cắt tỉaAlpha-Beta hoạt động bằng cách duy trì hai giá trị trong quá trình duyệt cây:

**Alpha (α)**: Giá trị tốt nhất (cao nhất) mà người chơi MAX đã tìm thấy cho đến nay trên đường đi từ gốc đến nút hiện tại. Nó đại diện cho điểm số tối thiểu mà MAX có thể đảm bảo nhận được. Ban đầu được khởi tạo là -∞ (âm vô cực).

**Beta (β)**: Giá trị tốt nhất (thấp nhất) mà người chơi MIN đã tìm thấy cho đến nay trên đường đi từ gốc đến nút hiện tại. Nó đại diện cho điểm số tối đa mà MIN sẽ cho phép MAX nhận được nếu MIN chơi tối ưu. Ban đầu được khởi tạo là +∞ (dương vô cực).

**Nguyên tắc cắt tỉa:**

Nếu tại bất kỳ nút nào trong quá trình tìm kiếm, điều kiện β <= α trở thành đúng, thì nhánh cây bắt đầu từ nút đó có thể được cắt tỉa (không cần duyệt thêm). Có hai loại cắt tỉa:

**Cắt tỉa Alpha** (Alpha Cutoff - xảy ra tại nút MIN): Nếu giá trị β hiện tại của nút MIN (điểm số thấp nhất có thể ép đối phương nhận trong nhánh này) trở nên nhỏ hơn hoặc bằng giá trị α mà nó kế thừa từ nút MAX cha (điểm số tối thiểu mà MAX đã được đảm bảo ở một nhánh khác), thì nút MIN và các nhánh con còn lại có thể bị cắt tỉa.

**Cắt tỉa Beta** (Beta Cutoff - xảy ra tại nút MAX): Nếu giá trị α hiện tại của nút MAX (điểm số cao nhất nó có thể đạt được trong nhánh này) trở nên lớn hơn hoặc bằng giá trị β mà nó kế thừa từ nút MIN cha (điểm số tối đa mà MIN sẽ cho phép), thì nút MAX và các con còn lại của nó có thể bị cắt tỉa.

Bắt đầu

function findBestMove(root\_state, max\_depth):

best\_value = alphabeta(root\_state, max\_depth, -infinity, +infinity, true)

function alphabeta(current\_state, depth, alpha, beta, is\_maximizing\_player):

if depth == 0 or is\_terminal(current\_state):

return evaluate\_heuristic(current\_state)

if is\_maximizing\_player:

max\_eval = -infinity

for each next\_state achievable from current\_state:

eval = alphabeta(next\_state, depth - 1, alpha, beta, false)

max\_eval = max(max\_eval, eval)

alpha = max(alpha, eval)

if beta <= alpha:

break

return max\_eval

else:

min\_eval = +infinity

for each next\_state achievable from current\_state:

eval = alphabeta(next\_state, depth - 1, alpha, beta, true)

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

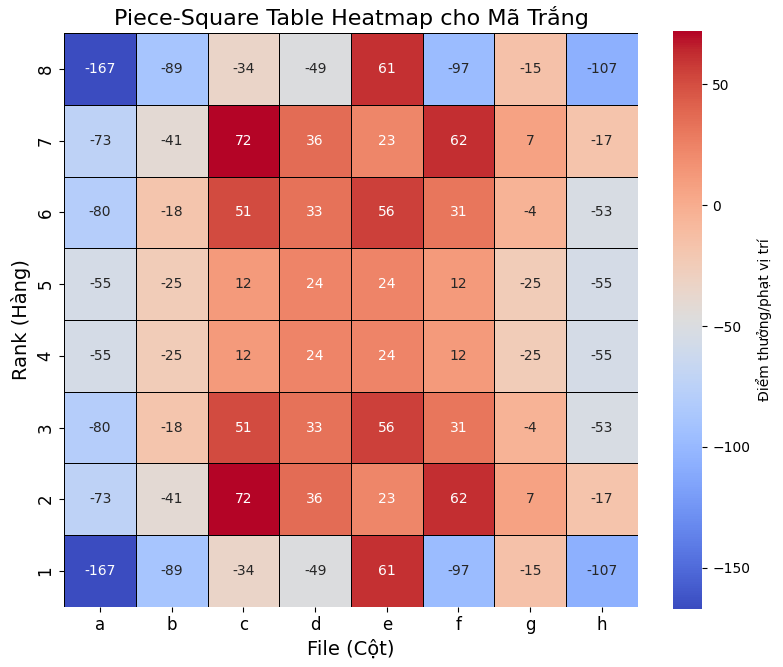
break

return min\_eval

Kết thúc

**Các biến có trong giả mã:**

* *root\_state*: Được truyền vào làm trạng thái bắt đầu cho hàm đệ quy *alphabeta*.
* *max\_depth*: Được truyền vào làm độ sâu ban đầu cho hàm *alphabeta*.
* *best\_value*: Nhận giá trị trả về từ *alphabeta*. Giá trị này cho biết kết quả dự kiến nếu cả hai người chơi đều chơi tối ưu.
* *best\_move*: Lưu trữ nước đi cụ thể từ *root\_state* dẫn đến *best\_value*.
* *current\_state*: Trạng thái trò chơi đang được đánh giá trong lần gọi hàm đệ quy này.
* *depth*: Kiểm tra trong điều kiện dừng (*depth == 0*). Giảm đi 1 khi gọi đệ quy *alphabeta*(..., *depth - 1*, ...).
* *alpha*: Đại diện cho giá trị tối thiểu mà người chơi MAX có thể đảm bảo nhận được trên đường đi từ gốc đến *current\_state*. Hoạt động như một giới hạn dưới cho điểm số cuối cùng của MAX.
* *beta*: Đại diện cho giá trị tối đa mà người chơi MIN sẽ cho phép MAX nhận được trên đường đi từ gốc đến *current\_state*. Hoạt động như một giới hạn trên cho điểm số cuối cùng của MAX.
* *is\_maximizing\_player*: Điều khiển việc thực thi nhánh *if* (MAX) hay *else* (MIN). Giá trị này được đảo ngược khi gọi đệ quy.
* *max\_eval*:Biến cục bộ chỉ dùng trong nhánh MAX. Lưu trữ giá trị cao nhất tìm thấy được trong số các *next\_state* đã được duyệt của *current\_state* này.
* *next\_state*: Biến cục bộ trong vòng lặp. Đại diện cho một trạng thái có thể đạt được sau một nước đi từ *current\_state*.
* *eval*: Biến cục bộ. Lưu trữ tạm thời kết quả trả về từ một lệnh gọi đệ quy *alphabeta* trên một *next\_state*. Được dùng để cập nhật *max\_eval* (hoặc *min\_eval*) và cập nhật *alpha* (hoặc *beta*).
* *min\_eval*: Biến cục bộ chỉ dùng trong nhánh MIN. Lưu trữ giá trị thấp nhất tìm thấy được trong số các *next\_state* đã được duyệt của *current\_state* này.
  + 1. ***Heatmap cho thuật toán Piece-Square Table***



Hình 13. Heatmap cho thuật toán Piece-Square Table cho Mã Trắng

Trong đó:

* Lưới 8x8 là đại diện cho bàn cờ vua, các ô tương ứng với mỗi ô trên bàn cờ.
* Các giá trị trong ô là điểm PST cho những ô đó
  + Số dương: Ô đó được coi là vị trí tốt cho Mã Trắng, phần mềm sẽ cộng điểm thưởng này, tương ứng với màu đỏ, vàng, cam.
  + Số âm: Ô đó được coi là vị trí xấu cho Mã Trắng, phần mềm sẽ trừ điểm, tương ứng với màu xanh dương đậm.
  + Số 0 hoặc gần 0: Ô trung tính, không thưởng, không phạt, tương ứng với các màu còn lại.
    1. ***Giả mã cho giải thuật Piece-Square Table***

Một cách đơn giản để gán giá trị cho các quân cờ cụ thể trên các ô vuông cụ thể. Một bảng được tạo cho mỗi quân cờ của mỗi màu và các giá trị được gán cho mỗi ô vuông. Sơ đồ này nhanh, vì thuật ngữ đánh giá từ các bảng ô vuông có thể được cập nhật gia tăng khi các nước đi được thực hiện và hủy bỏ trong cây tìm kiếm. Do tốc độ đó, các bảng ô vuông có ích rất nhiều khi tiến hành Lazy evaluation. (Schröder, 2002)

Bắt đầu

function evaluate\_board(board\_state):

total\_score = 0

pst\_score = 0

for square\_index from 0 to 63:

piece = get\_piece\_at(board\_state, square\_index)

if piece is not EMPTY:

piece\_type = get\_piece\_type(piece)

piece\_color = get\_piece\_color(piece)

selected\_pst = get\_pst\_table(piece\_type)

rank = calculate\_rank(square\_index)

file = calculate\_file(square\_index)

if piece\_color is BLACK:

lookup\_rank = 7 - rank

else:

lookup\_rank = rank

lookup\_file = file

positional\_value = selected\_pst[lookup\_rank][lookup\_file]

if piece\_color is WHITE:

pst\_score = pst\_score + positional\_value

else:

pst\_score = pst\_score - positional\_value

total\_score = total\_score + pst\_score

return total\_score

Kết thúc

**Các biến có trong giả mã:**

* *evaluate\_board(board\_state)*: Hàm chính, tính toán điểm số đánh giá tổng thể cho một trạng thái bàn cờ (*board\_state*) cụ thể.
* *get\_piece\_at(board\_state, square\_index)*: Lấy thông tin về quân cờ hoặc ô trống tại một vị trí cụ thể trên bàn cờ.
* *get\_piece\_type(piece)*: Xác định loại của một quân cờ (không phân biệt màu).
* *get\_piece\_color(piece)*: Xác định màu của một quân cờ.
* *get\_pst\_table(piece\_type)*: Tra cứu trong một tập hợp các bảng PST đã được định nghĩa sẵn dựa trên *piece\_type*.
* *calculate\_rank(square\_index)*: Chuyển đổi chỉ số ô 1 chiều (0-63) thành chỉ số hàng 2 chiều (0-7), thường dùng phép chia lấy nguyên: *rank = square\_index // 8*.
* *calculate\_file(square\_index)*: Chuyển đổi chỉ số ô 1 chiều (0-63) thành chỉ số cột 2 chiều (0-7), thường dùng phép chia lấy dư: *rank = square\_index % 8*.
  1. **Mô tả bài toán**

Cờ vua từ lâu đã được coi là thước đo trí tuệ và tư duy chiến lược của con người. Tuy nhiên, với sự phát triển của trí tuệ nhân tạo (AI), cuộc chơi đã thay đổi đáng kể. Từ những hệ thống đơn giản như chương trình đánh cờ đầu tiên vào những năm 1950 đến sự xuất hiện của Deep Blue – cỗ máy đánh bại đại kiện tướng Garry Kasparov vào năm 1997, AI trong cờ vua đã tiến một bước dài. Ngày nay, các thuật toán tiên tiến như AlphaZero của DeepMind có thể tự học chơi cờ mà không cần dữ liệu huấn luyện từ con người, đạt đến trình độ siêu việt chỉ sau vài giờ tự rèn luyện.

Bài toán AI trong cờ vua có thể được xem là một bài toán ra quyết định trong môi trường có thông tin đầy đủ nơi hai tác nhân đối kháng luân phiên thực hiện nước đi với mục tiêu tối ưu hóa lợi ích của mình. Do không có yếu tố ngẫu nhiên và tất cả thông tin về trạng thái trò chơi đều được công khai, bài toán này là một thử thách lớn cho việc tìm kiếm trạng thái kết hợp với học máy.

Bài toán có thể được mô tả dưới dạng hình thức như sau:

* Tập trạng thái ***S***: Gồm tất cả các cấu hình hợp lệ của bàn cờ tại một thời điểm nhất định.
* Tập hành động ***A(s)***: Tập hợp tất cả các nước đi có thể thực hiện được từ trạng thái ss.
* Hàm chuyển trạng thái ***T(s,a)***: Một hàm ánh xạ từ trạng thái hiện tại và nước đi thành trạng thái mới.
* Hàm đánh giá ***V(s)***: Một hàm ánh xạ trạng thái bàn cờ thành một giá trị số phản ánh mức độ có lợi của trạng thái đó đối với người chơi.
* Chính sách ***π(s)***: Một hàm ánh xạ trạng thái hiện tại đến hành động tối ưu tương ứng.
* Hàm phần thưởng ***R(s)***: Một hàm cho biết điểm số hoặc lợi ích của trạng thái ss, thường được xác định theo kết quả của ván đấu (thắng, thua hoặc hòa).

Mục tiêu của hệ thống AI là tìm ra một chính sách **π(s)** tối ưu sao cho nó dẫn đến kết quả có lợi nhất trong dài hạn.

* 1. **Phân tích nghiệp vụ bài toán**

Trong quá trình xây dựng một hệ thống trí tuệ nhân tạo có khả năng chơi cờ vua, việc phân tích chính xác các nghiệp vụ cốt lõi đóng vai trò quyết định đến chất lượng và hiệu suất của sản phẩm cuối cùng. Qua quá trình nghiên cứu và đánh giá, chúng tôi đã xác định được hai nhóm nghiệp vụ chính cần được ưu tiên triển khai. Mỗi nghiệp vụ này đều sở hữu những đặc thù riêng và đòi hỏi những giải pháp kỹ thuật khác biệt để đạt được hiệu quả tối ưu.

* + 1. ***Đánh giá trạng thái bàn cờ.***
* Nghiệp vụ đánh giá trạng thái bàn cờ đòi hỏi sự phân tích chi tiết về tình huống hiện tại của ván đấu để từ đó đưa ra những nhận định chính xác về lợi thế tương đối giữa hai bên. Quá trình này cần tích hợp nhiều yếu tố chiến thuật và chiến lược khác nhau. Cụ thể, chúng tôi tập trung vào việc xây dựng một hàm đánh giá toàn diện, có khả năng xem xét đồng thời nhiều khía cạnh của bàn cờ như vị trí các quân, cấu trúc tốt, an toàn của vua, và kiểm soát trung tâm. Hệ thống này sẽ được thiết kế để lượng hóa các yếu tố trên thành các giá trị số học, tạo điều kiện cho việc so sánh và lựa chọn nước đi tối ưu.
* Ngoài ra, việc áp dụng các kỹ thuật học sâu trong quá trình đánh giá trạng thái cũng mang lại nhiều triển vọng đáng kể. Các mô hình học sâu có khả năng nhận diện những mẫu vị trí phức tạp mà các phương pháp truyền thống khó có thể phát hiện. Điều này cho phép hệ thống đưa ra những đánh giá chính xác hơn, đặc biệt trong các tình huống phức tạp hoặc không rõ ràng.
  + 1. ***Tìm kiếm và ra quyết định.***
* Nghiệp vụ thứ hai, không kém phần quan trọng, liên quan đến quá trình tìm kiếm không gian trạng thái và ra quyết định lựa chọn nước đi. Trong bối cảnh của trò chơi cờ vua, không gian trạng thái có kích thước vô cùng lớn, đòi hỏi những phương pháp tìm kiếm thông minh và hiệu quả. Thuật toán Minimax với cơ chế cắt tỉa Alpha-Beta là lựa chọn ưu việt cho vấn đề này, cho phép hệ thống khám phá không gian trạng thái một cách có chiến lược, tránh việc kiểm tra những nhánh không cần thiết trong cây tìm kiếm.
* Bên cạnh đó, việc tích hợp giải thuật suy nghiệm Piece-Square Table mang lại một lợi thế không nhỏ. Kỹ thuật này cung cấp thông tin heuristic quý giá về giá trị tương đối của mỗi quân cờ tại từng vị trí cụ thể trên bàn cờ. Các bảng giá trị này được thiết kế dựa trên kinh nghiệm và kiến thức chuyên môn về cờ vua, giúp định hướng quá trình tìm kiếm theo hướng có lợi về mặt chiến lược. Nhờ đó, hệ thống có thể giảm đáng kể thời gian xử lý, đồng thời nâng cao chất lượng của các quyết định được đưa ra.
  1. **mô hình, quy trình nghiệp vụ bài toán**
* Mô hình đề xuất cho hệ thống AI chơi cờ vua của chúng tôi được xây dựng trên nền tảng kết hợp giữa các thuật toán tìm kiếm truyền thống và các kỹ thuật heuristic hiện đại. Cụ thể, chúng tôi đề xuất một kiến trúc đa tầng, trong đó thuật toán Minimax với cơ chế cắt tỉa Alpha-Beta đóng vai trò là hạt nhân của quy trình tìm kiếm, trong khi các giải thuật suy nghiệm như Piece-Square Table cung cấp thông tin bổ sung để định hướng và tối ưu hóa quá trình này.
* Quy trình nghiệp vụ của hệ thống bắt đầu từ việc phân tích trạng thái hiện tại của bàn cờ. Tại mỗi lượt đi, hệ thống sẽ xây dựng một cây tìm kiếm các nước đi có thể, sau đó áp dụng thuật toán Minimax để đánh giá các nhánh của cây này. Cơ chế cắt tỉa Alpha-Beta được sử dụng trong quá trình này để loại bỏ những nhánh chắc chắn không dẫn đến kết quả tối ưu, giúp giảm đáng kể khối lượng tính toán cần thực hiện.
* Mô-đun tìm kiếm của hệ thống đóng vai trò quan trọng trong việc khám phá không gian trạng thái một cách hiệu quả. Nó sử dụng thuật toán Minimax làm nền tảng, kết hợp với các kỹ thuật tối ưu hóa như Alpha-Beta Pruning để giảm thiểu số lượng trạng thái cần kiểm tra. Ngoài ra, các giải thuật suy nghiệm như Piece-Square Table cũng được tích hợp để cung cấp thông tin bổ sung về giá trị chiến lược của từng vị trí, góp phần định hướng quá trình tìm kiếm theo hướng có lợi hơn.
* Một đặc điểm nổi bật của mô hình đề xuất là khả năng điều chỉnh độ sâu tìm kiếm dựa trên độ phức tạp của vị trí và thời gian suy nghĩ khả dụng. Điều này cho phép hệ thống phân bổ tài nguyên tính toán một cách thông minh, tập trung vào việc phân tích kỹ lưỡng những tình huống phức tạp hoặc quan trọng về mặt chiến thuật. Các vị trí đơn giản hơn, như những tình huống có lợi thế vật chất rõ ràng, có thể được xử lý nhanh chóng với độ sâu tìm kiếm thấp hơn.
* Việc áp dụng các kỹ thuật như:
  + Sắp xếp nước đi để ưu tiên kiểm tra những nước đi có tiềm năng cao trước
  + Cơ chế mở rộng có chọn lọc cho phép hệ thống tập trung vào những nhánh tìm kiếm hứa hẹn

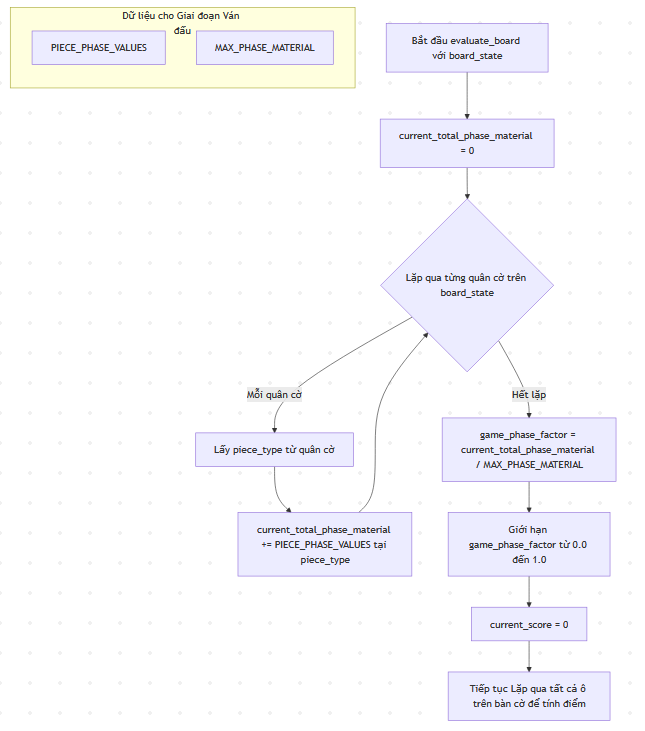
Tất cả những yếu tố trên kết hợp lại tạo nên một mô hình toàn diện và hiệu quả cho việc phát triển một hệ thống AI có khả năng chơi cờ vua ở trình độ cao.

# CHƯƠNG 3: Phương pháp/Kỹ thuật

# Phương pháp

## Mô hình giải quyết bài toán

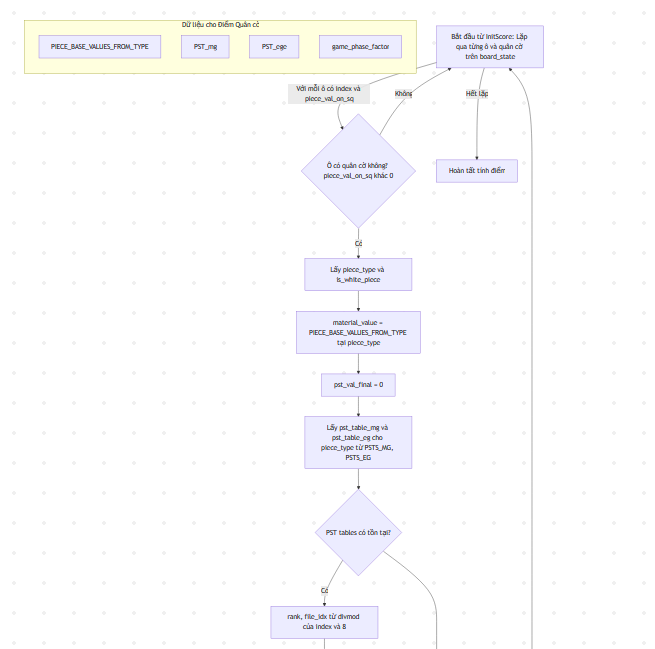
* Đánh giá trạng thái bàn cờ.
  + Hàm chịu trách nhiệm tính toán điểm số cho một thế cờ nhất định dựa trên các giá trị PIECE\_PHASE\_VALUES/MAX\_PHASE\_MATERIAL để đánh giá và xác định xem bàn cờ đang ở giai đoạn khai cuộc/trung cuộc (giá trị 1) hay tàn cuộc (giá trị 0) và sử dụng các giá trị trên bảng các Pieces-Square Tables (PST) được định sẵn: PSTS\_MG cho trung cuộc và PSTS\_EG cho tàn cuộc

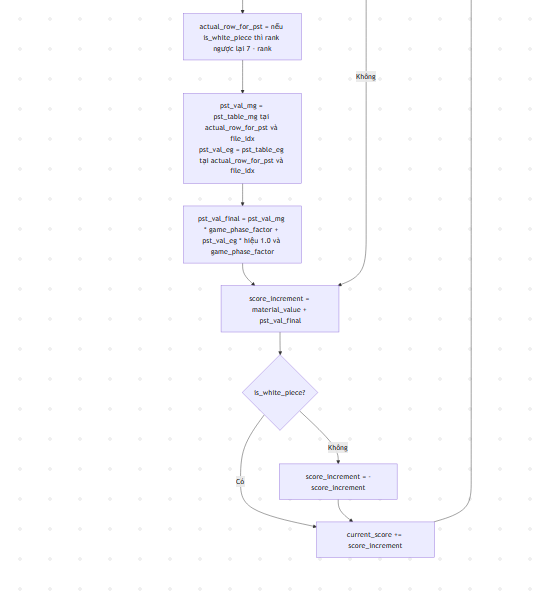


Hình 14. Sơ đồ mô tả thuật toán của phần mềm cờ vua (chức năng evaluate\_board).

* + Mỗi loại quân cờ (Tốt, Mã, Tượng, Xe, Hậu, Vua) có một bảng PST riêng cho MG và EG. Bảng này gán một điểm thưởng/phạt cho quân cờ đó tùy thuộc vào vị trí nó đứng trên bàn cờ. Ví dụ, một quân Mã ở trung tâm thường có giá trị vị trí cao hơn một quân Mã ở góc.
  + pst\_val\_final được tính bằng cách nội suy giữa giá trị PST trung cuộc và giá trị PST tàn cuộc, dựa trên game\_phase\_factor đã tính ở trên:

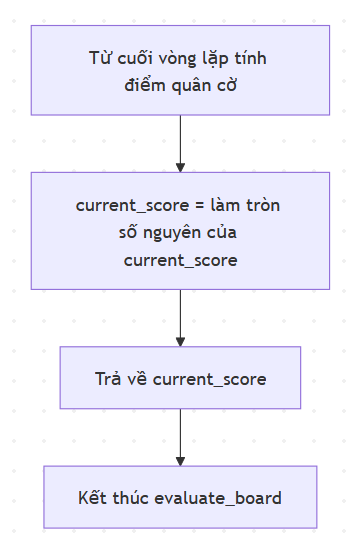
***pst\_final = pst\_ mg \* game\_phase\_factor + pst\_ eg \* (1 - game\_phase\_factor)***





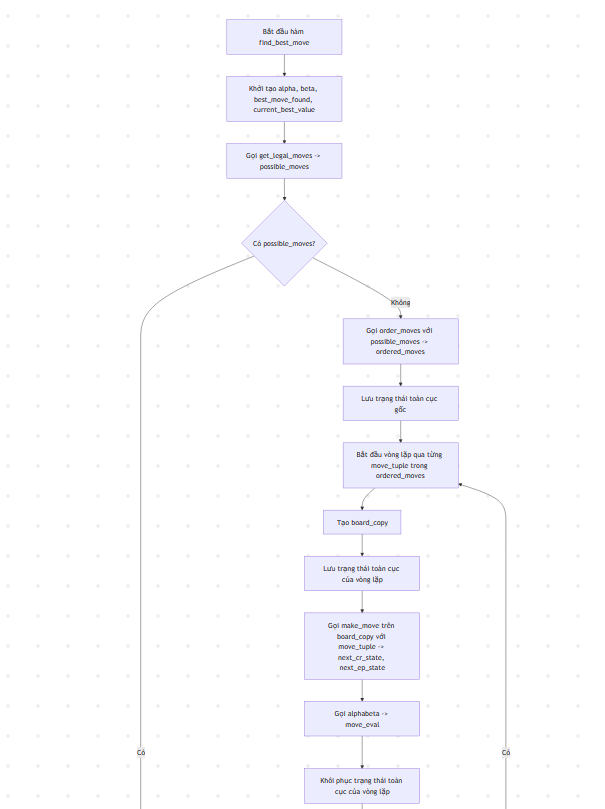
Hình 15. Sơ đồ evaluate\_board - lặp qua ô và tính điểm từng quân cờ

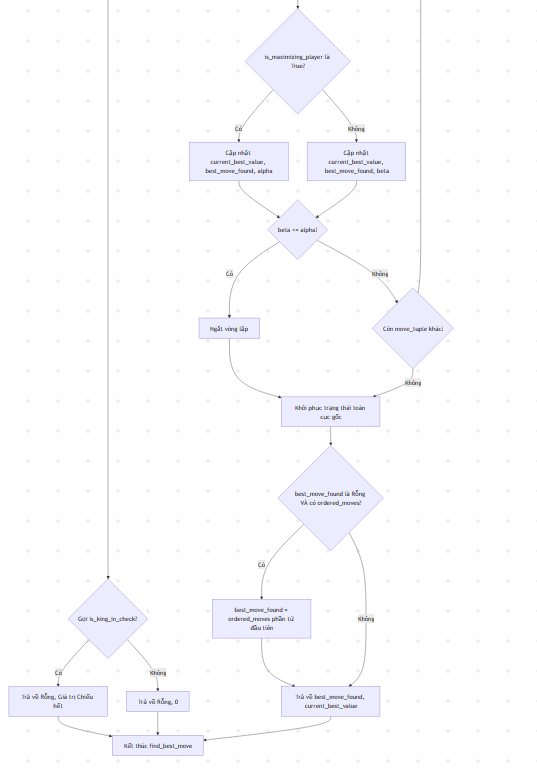
* Điểm của quân cờ hiện tại (*material\_value* + *pst \_final*) được cộng vào tổng điểm (*current\_score*). Nếu quân cờ là của phe Đen, điểm này sẽ bị trừ đi.
* **Kết quả:** Hàm trả về *current\_score* (đã làm tròn), với số dương lợi cho Trắng, số âm lợi cho Đen.



Hình 16. Sơ đồ evaluate\_board - hoàn tất và trả về điểm số

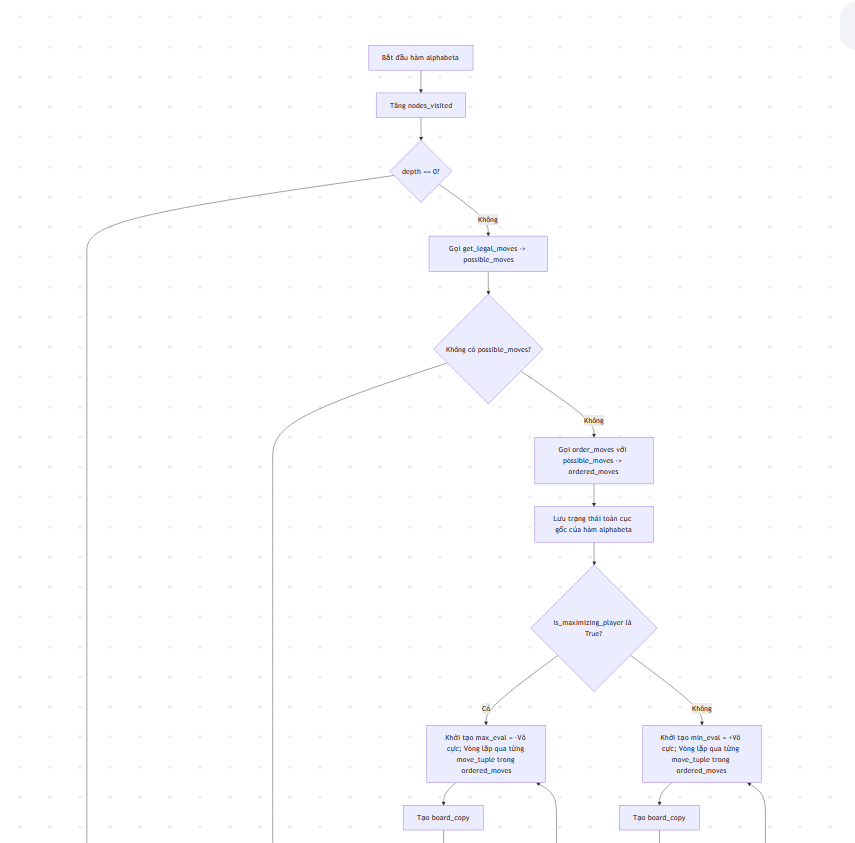
* **Các Cấu trúc Dữ liệu Hỗ trợ Đánh giá:**
* PIECE\_BASE\_VALUES\_FROM\_TYPE: Lưu giá trị cơ bản (vật chất) của từng loại quân cờ.
* PIECE\_PHASE\_VALUES và MAX\_PHASE\_MATERIAL: Dùng để tính toán giai đoạn của ván cờ.
* pawn\_pst\_mg, knight\_pst\_mg, ... king\_pst\_mg: Các mảng 2D chứa bảng điểm vị trí cho từng loại quân cờ trong giai đoạn trung cuộc.
* pawn\_pst\_eg, knight\_pst\_eg, ... king\_pst\_eg: Tương tự, nhưng cho giai đoạn tàn cuộc.
* PSTS\_MG và PSTS\_EG: Các dictionary ánh xạ loại quân cờ tới bảng PST tương ứng (MG hoặc EG) để dễ truy cập.
* Tìm kiếm và ra quyết định.
  + Hàm *make\_move* là một trong phần quan trọng nhất của phần mềm cờ vua này, hàm này thực hiện các nước đi trên bàn cờ và các logic đặc biệt như bắt Tốt qua đường, phong cấp và nhập thành,.
  + Hàm *find\_best\_move* sử dụng thuật toán Minimax kết hợp với cắt tỉa Alpha-Beta để tìm nước đi tối ưu. Bộ giải sẽ đánh giá tất cả các nước đi hợp lệ từ trạng thái hiện tại, duyệt qua một cây trò chơi theo độ sâu được chỉ định, và lựa chọn nước đi có điểm số tốt nhất theo lượt chơi (MAX hoặc MIN).

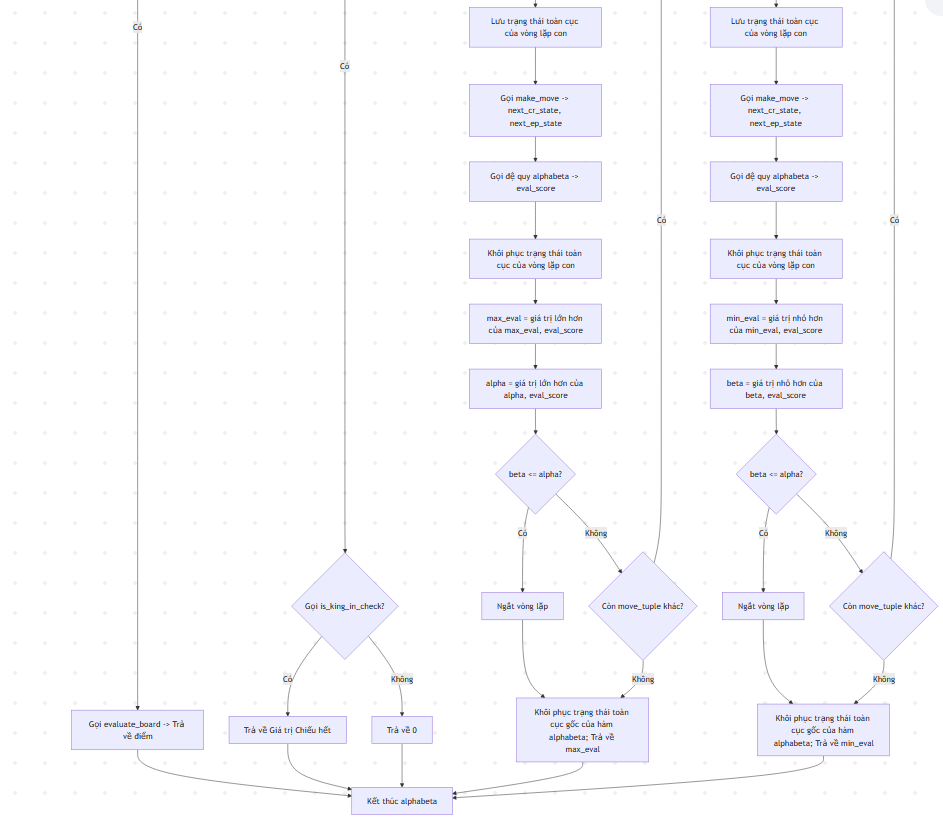




Hình 17. Sơ đồ hàm make\_move

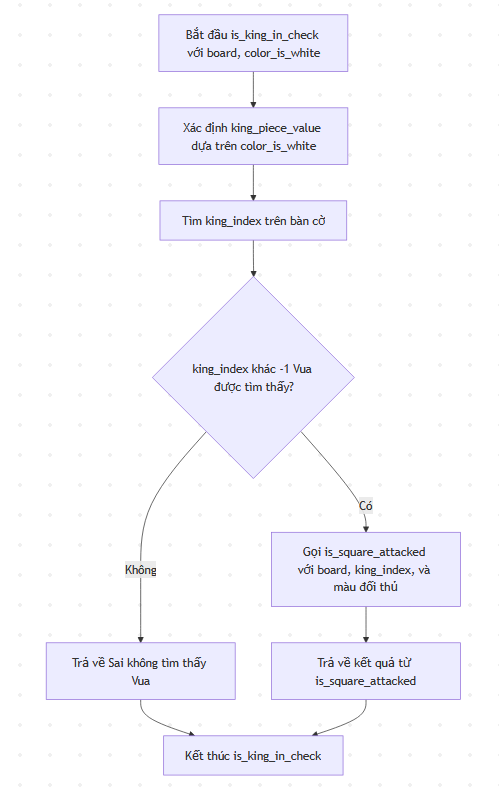
* **Cắt tỉa Alpha-Beta (*alphabeta*)**: Là phiên bản tối ưu hóa của Minimax, kỹ thuật này loại bỏ các nhánh không cần thiết trong cây tìm kiếm khi nhận thấy rằng không thể có kết quả tốt hơn các nhánh đã duyệt. Nhờ đó, hiệu suất tìm kiếm được tăng lên đáng kể.
* Trong hàm Alpha-Beta, điều kiện dừng đệ quy (Depth == 0): Khi độ sâu tìm kiếm đạt đến 0 trong hàm *alphabeta*, nó sẽ gọi *evaluate\_board(board\_state)* để lấy điểm số cho thế cờ lá đó.
* **Xử lý thế cờ kết thúc (Checkmate/Stalemate):**
  + Khi không còn nước đi hợp lệ (*if not possible\_moves:*), các hàm này sẽ kiểm tra xem có phải là Chiếu Bí (*is\_king\_in\_check*) hay Hòa Cờ.
  + Nếu Chiếu Bí: Trả về giá trị vô cực (-float('inf') hoặc float('inf')) tùy thuộc vào phe nào bị chiếu bí. Đây là một dạng đánh giá trạng thái cuối cùng.
  + Nếu Hòa Cờ: Trả về 0. Đây cũng là một dạng đánh giá trạng thái cuối cùng.
* **Sắp xếp nước đi (*order\_moves*)**: Để nâng cao hiệu quả của thuật toán Alpha-Beta, các nước đi được sắp xếp theo thứ tự ưu tiên, ví dụ: nước ăn quân giá trị cao, nước phong hậu,... Điều này giúp cắt tỉa sớm hơn trong quá trình tìm kiếm.
* **Sinh nước đi (*get\_legal\_moves*)**: Tất cả các nước đi hợp lệ được tạo ra bằng cách lọc từ danh sách nước đi "giả hợp lệ" (pseudo-legal) – tức các nước đi hợp lệ về mặt kỹ thuật nhưng có thể để lộ vua – để loại bỏ các nước khiến quân Vua bị chiếu.





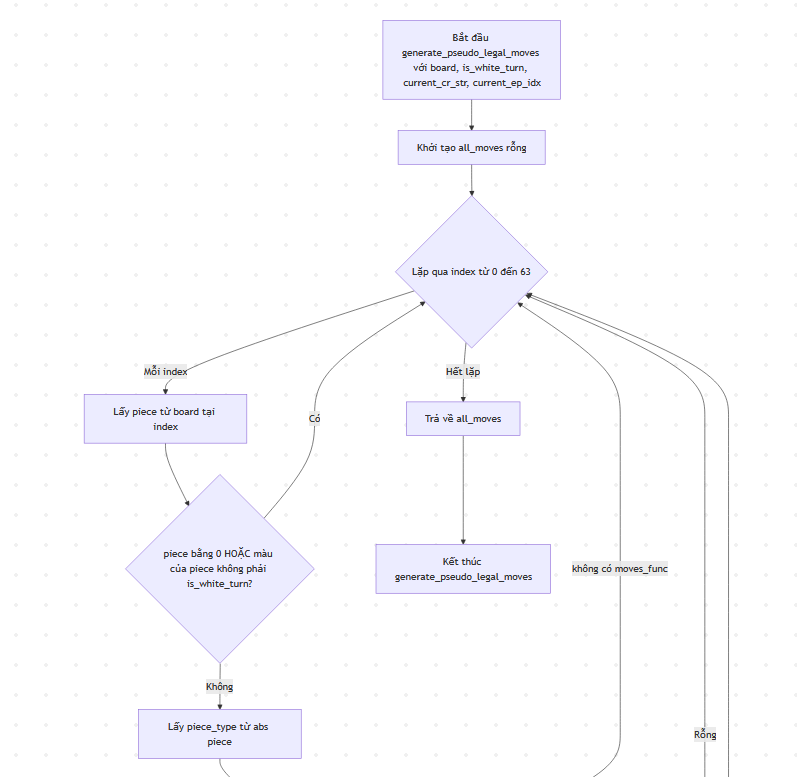
Hình 18. Sơ đồ hàm alphabeta

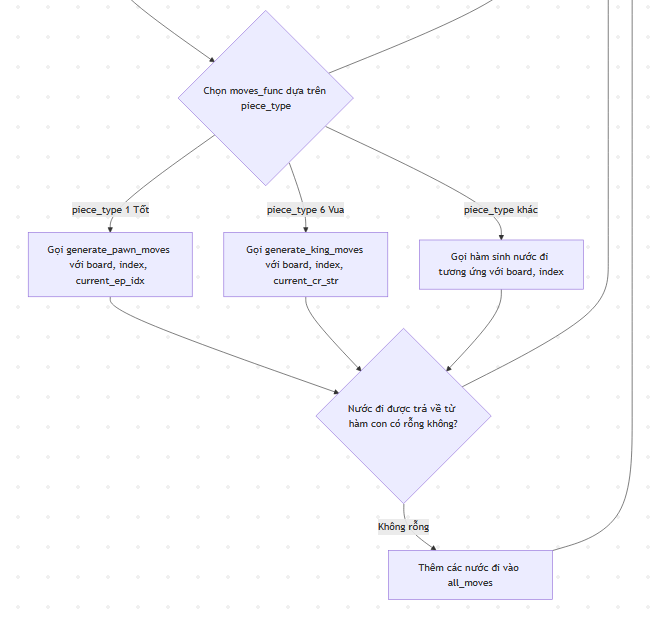
* Hàm is\_king\_in\_check là một vòng lặp for kiểm tra 64 ô cờ liên tục để kiểm tra xem Vua trắng/đen có đang bị chiếu không, phụ thuộc vào hàm is\_square\_attacked:



Hình 19. Hàm is\_king\_in\_check

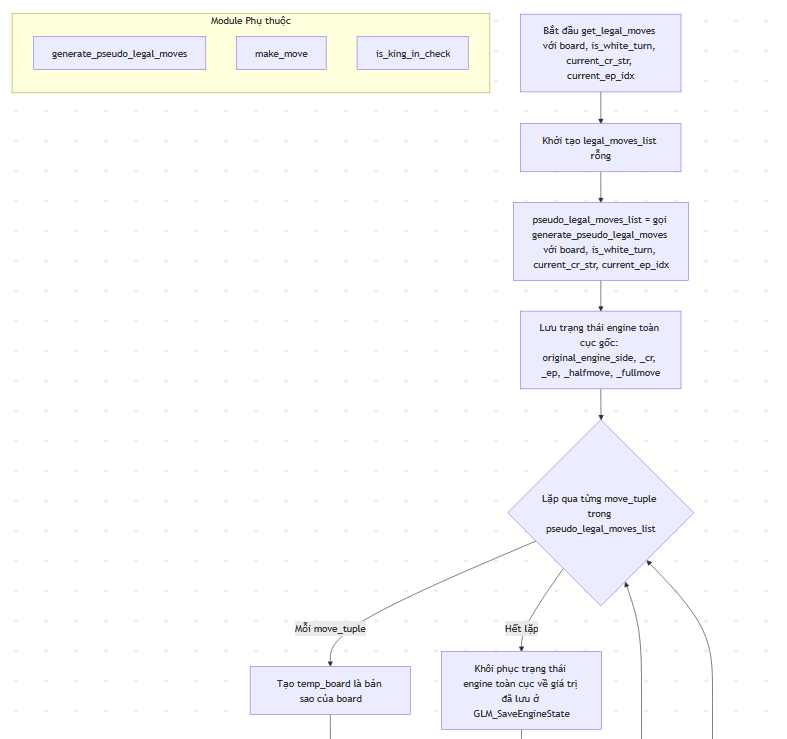
* Hàm get\_legal\_moves trả về một danh sách các nước đi hợp lệ cho người chơi hiện tại bằng cách:
  + Gọi hàm *generate\_pseudo\_legal\_moves* để tạo ra tất cả các nước đi có thể theo quy tắc di chuyển của từng quân cờ. Tuy nhiên, các nước đi này chưa được kiểm tra xem có để vua bị chiếu hay không.

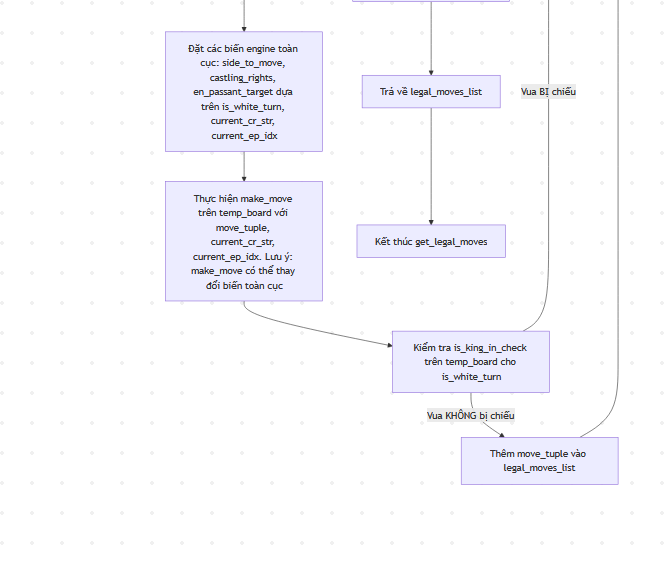




Hình 20. Sơ đồ hàm generate\_pseudo\_legal\_moves

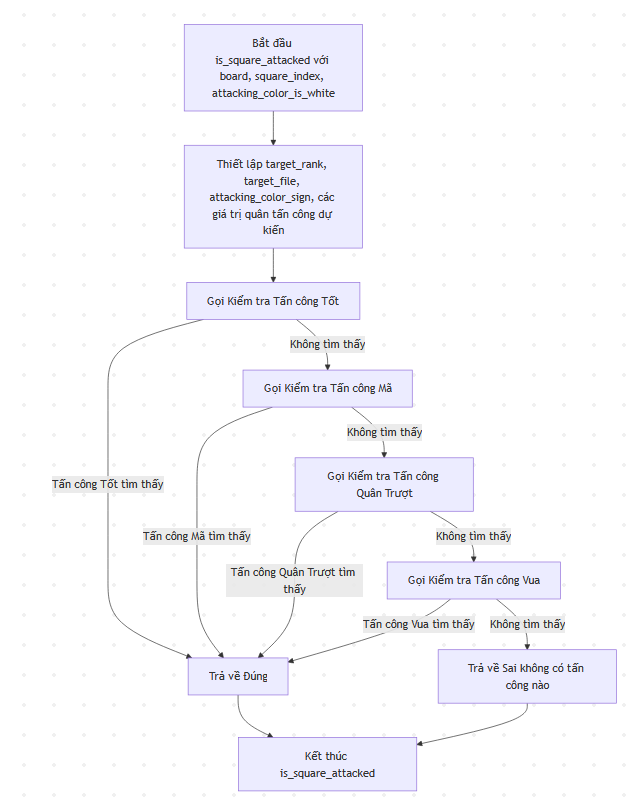
* + Kiểm tra tính hợp lệ của từng nước đi, với mỗi nước đi trong *pseudo\_legal\_moves* tạo một bản sao của bàn cờ (*temp\_board*) để thử nghiệm nước đi.
  + Gọi hàm *make\_move* để thực hiện nước đi trên bản sao bàn cờ. Hàm này cũng trả về thông tin về quân cờ bị bắt, quyền nhập thành cũ, và mục tiêu bắt tốt qua đường.
  + Kiểm tra xem sau khi thực hiện nước đi, vua của người chơi có bị chiếu hay không bằng cách *gọi is\_king\_in\_check*.
  + Nếu nước đi không khiến vua bị chiếu, thêm nó vào danh sách *legal\_moves*.
  + Sau khi kiểm tra tất cả các nước đi, trả về danh sách *legal\_moves*.





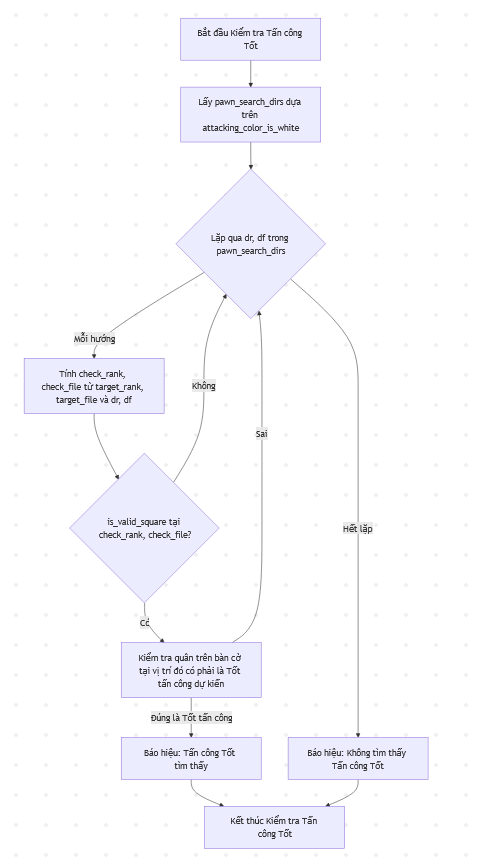
Hình 21. Hàm get\_legal\_moves

* Hàm is\_square\_attacked là một thành phần quan trọng trong engine cờ vua, có nhiệm vụ xác định xem một ô cụ thể trên bàn cờ (square\_index) có đang bị tấn công bởi bất kỳ quân cờ nào của một phe nhất định (attacking\_color\_is\_white) hay không. Đầu vào của hàm bao gồm trạng thái hiện tại của bàn cờ (board), chỉ số của ô cần kiểm tra (từ 0 đến 63), và một giá trị boolean cho biết phe đang được coi là tấn công là Trắng hay Đen.
* Đầu tiên, hàm thực hiện một số thiết lập cơ bản: nó chuyển đổi *square\_index* thành tọa độ hàng (*target\_rank*) và cột (*target\_file*) để dễ dàng tính toán. Đồng thời, nó xác định "dấu" của phe tấn công (*attacking\_color\_sign*, ví dụ: 1 cho Trắng, -1 cho Đen) và các giá trị số nguyên cụ thể đại diện cho quân Tốt, Mã, và Vua của phe tấn công trên mảng bàn cờ (**ví dụ**: nếu Trắng tấn công, Tốt tấn công dự kiến sẽ là 1\*1=1, Mã là 2\*1=2).
* Sau đó, hàm tiến hành kiểm tra tuần tự khả năng tấn công từ các loại quân khác nhau, và sẽ trả về True (Đúng) ngay khi phát hiện ra bất kỳ một mối đe dọa nào, mà không cần kiểm tra thêm các loại quân khác.



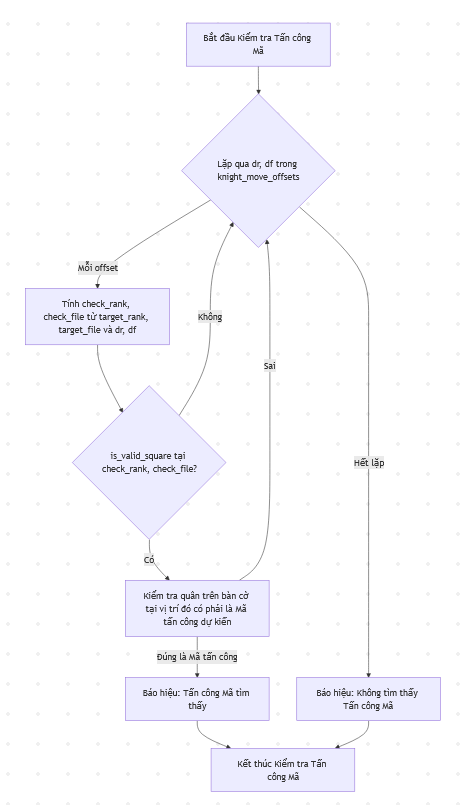
Hình 22. Sơ đồ hàm is\_square\_attacked (Luồng chính)

* **Kiểm tra Tấn công từ Tốt**: Hàm xác định các hướng tấn công tương đối của Tốt dựa trên màu sắc của phe tấn công (ví dụ: với hàng (r), cột (f), nếu Trắng tấn công ô ở (r,f), Tốt Đen có thể tấn công từ (r+1, f-1) hoặc (r+1, f+1)). Nó lặp qua các hướng này, tính toán vị trí của Tốt tấn công tiềm năng. Nếu vị trí đó hợp lệ trên bàn cờ và chứa đúng quân Tốt của phe tấn công, hàm kết luận ô đang bị tấn công bởi Tốt.



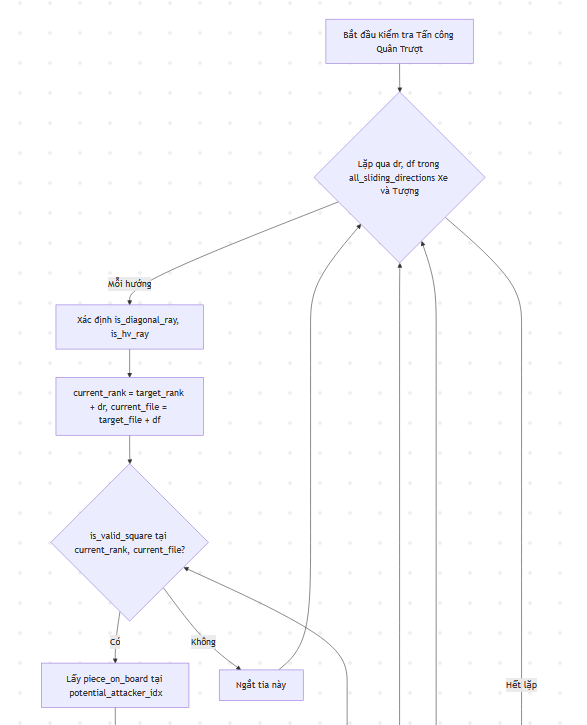
Hình 23. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công Tốt

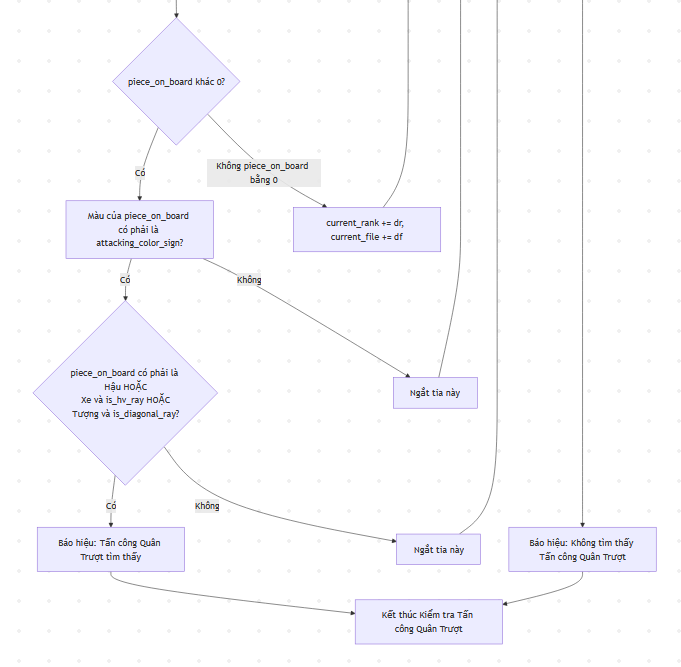
* **Kiểm tra Tấn công từ Mã**: Hàm lặp qua tất cả tám vị trí mà một quân Mã có thể nhảy tới từ ô mục tiêu (nhưng theo hướng ngược lại, tức là từ vị trí Mã có thể đứng để tấn công ô mục tiêu). Với mỗi vị trí tiềm năng của Mã, nếu nó hợp lệ và chứa đúng quân Mã của phe tấn công, ô được coi là bị tấn công.



Hình 24. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công Mã

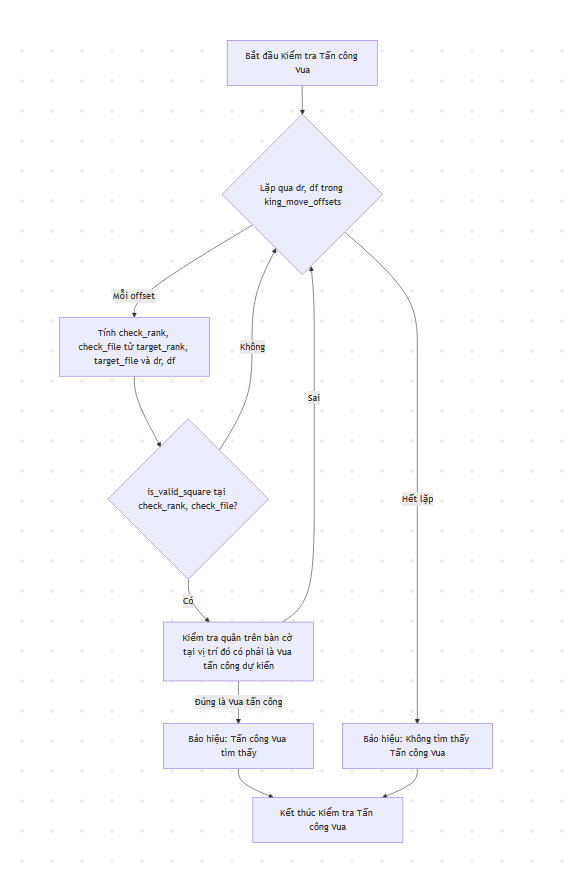
* **Kiểm tra Tấn công từ Quân Trượt (Xe, Tượng, Hậu)**: Hàm lặp qua tất cả tám hướng cơ bản (bốn hướng của Xe: ngang/dọc, và bốn hướng của Tượng: chéo). Với mỗi hướng:
  + Nó "trượt" ra xa từ square\_index theo hướng đó, từng ô một.
  + Tại mỗi ô trên đường trượt, nó kiểm tra quân cờ đang đứng ở đó.
  + Nếu ô đó trống, nó tiếp tục trượt xa hơn theo cùng một hướng.
  + Nếu gặp một quân cờ:
    - Nó kiểm tra xem quân cờ đó có thuộc phe tấn công hay không.
    - Nếu đúng là quân của phe tấn công, nó tiếp tục kiểm tra loại quân:
    - Nếu là Hậu, ô chắc chắn bị tấn công.
    - Nếu là Xe và hướng đang xét là ngang hoặc dọc, ô bị tấn công.
    - Nếu là Tượng và hướng đang xét là chéo, ô bị tấn công.
  + Trong bất kỳ trường hợp nào gặp quân cờ (dù cùng phe hay khác phe, dù có tấn công được hay không), việc "trượt" theo hướng đó sẽ dừng lại, vì quân cờ đó đã chặn đường đi hợp lệ.





Hình 25. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công quân trượt (Xe, Tượng, Hậu)

* **Kiểm tra Tấn công từ Vua**: Cuối cùng, hàm kiểm tra tám ô liền kề xung quanh square\_index. Nếu bất kỳ ô nào trong số đó chứa quân Vua của phe tấn công, ô được coi là bị tấn công (điều này quan trọng để xác định các nước đi hợp lệ cho Vua đối phương hoặc phát hiện chiếu).



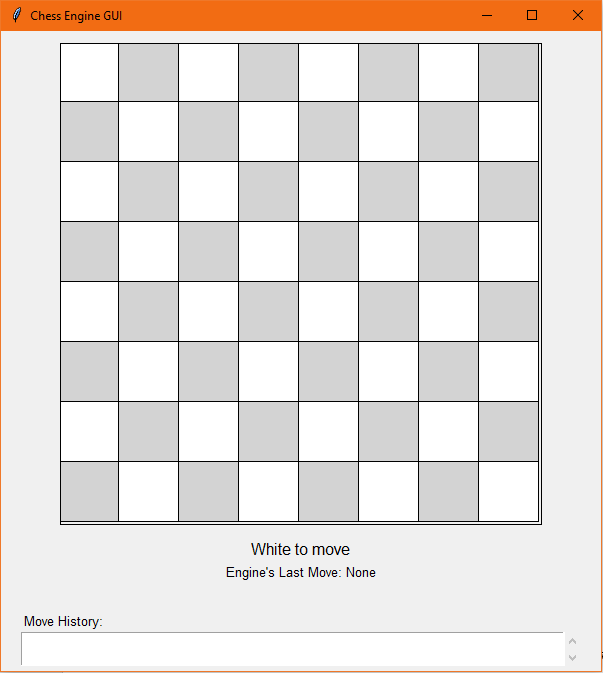
Hình 26. Sơ đồ hàm is\_square\_attacked - logic kiểm tra tấn công quân Vua

## Thiết kế bàn cờ vua

Trò chơi được thiết kế để chơi trên một bàn cờ có 8 hàng và 8 cột tạo thành tổng cộng 64 ô vuông, trong đó có 32 ô trắng và 32 ô đen.

Khung giao diện người dùng (GUI) có các thành phần chính như sau:

* Cửa sổ chính: chứa bàn cờ, quân cờ, lịch sử nước đi và các phần phụ trợ khác, có kích thước 600x640px.



Hình 27. Giao diện người dùng.

* Sử dụng hàm *draw\_board* để vẽ các ô cờ với màu sắc xen kẽ với nhau:

Hình 28. Hàm draw\_board,

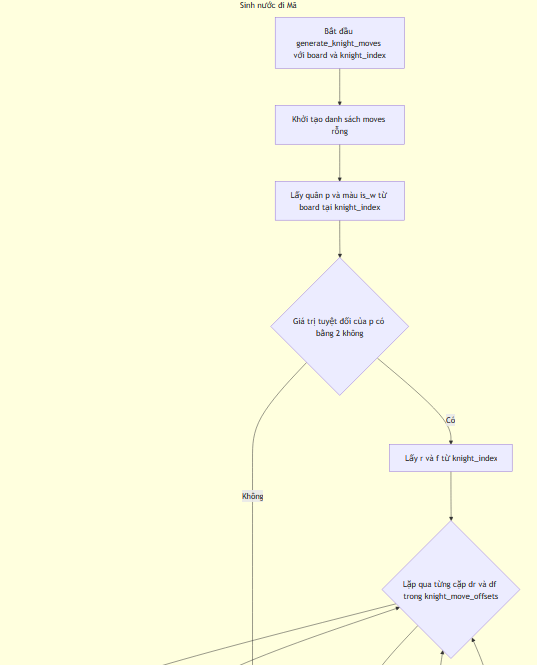
Bàn cờ có kích thước 480x480px với kích thước của các ô cờ bằng 480/8 = 60px.

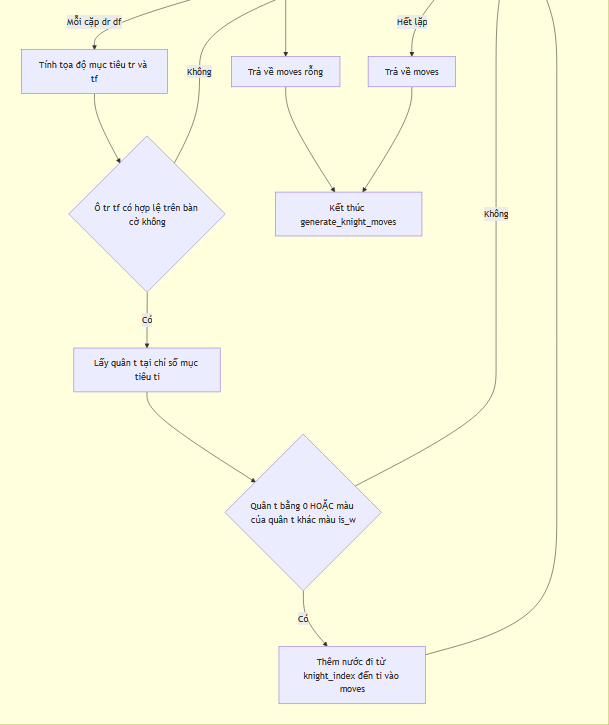
* Sử dụng hàm *load\_piece\_images* để tải các quân cờ từ file images có sẵn ảnh 12 quân cờ.
* Các quân cờ sẽ được sắp xếp theo dãy 1 chiều có 64 giá trị (index) từ 0-63 tượng trưng cho các ô vuông trên bàn cờ. Trong đó, các quân cờ được biểu thị theo các số tương ứng:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tốt | Mã | Tượng | Xe | Hậu | Vua |
| 1 | 2 | 3 | 4 | 5 | 6 |

Cuối cùng, ta có các hàm tạo nước đi cho từng quân:

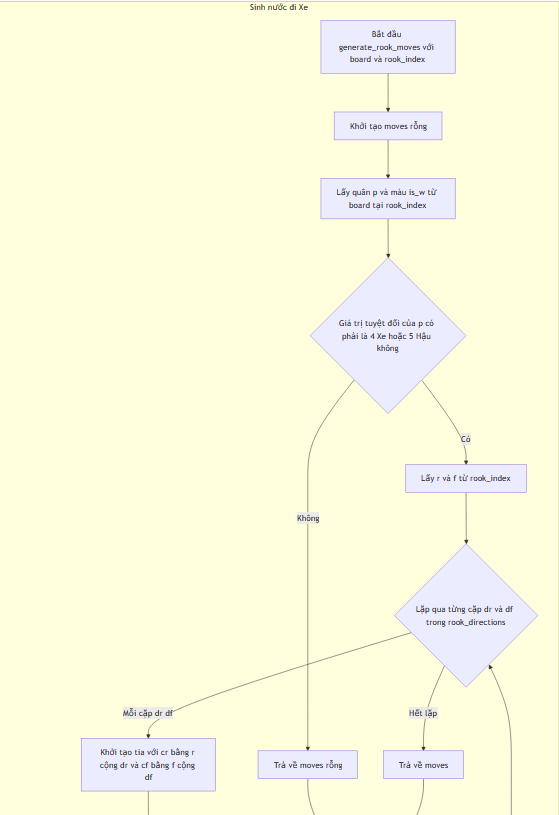
* + Quân Mã:

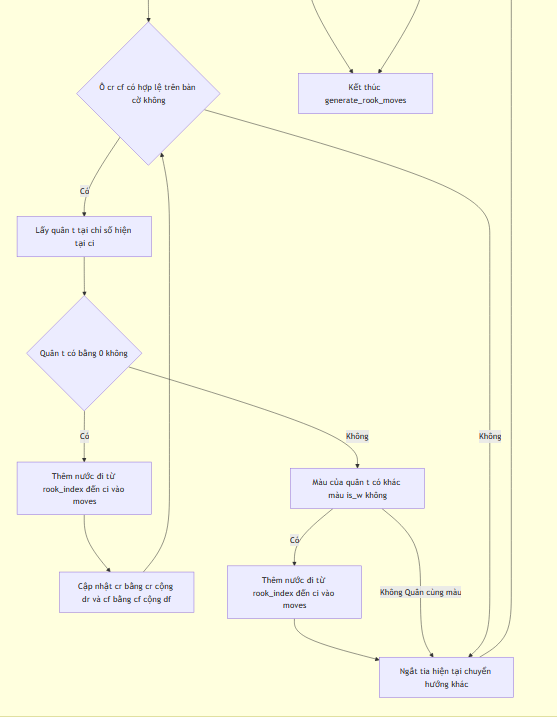




Hình 29. Sơ đồ hàm nước đi cho quân Mã.

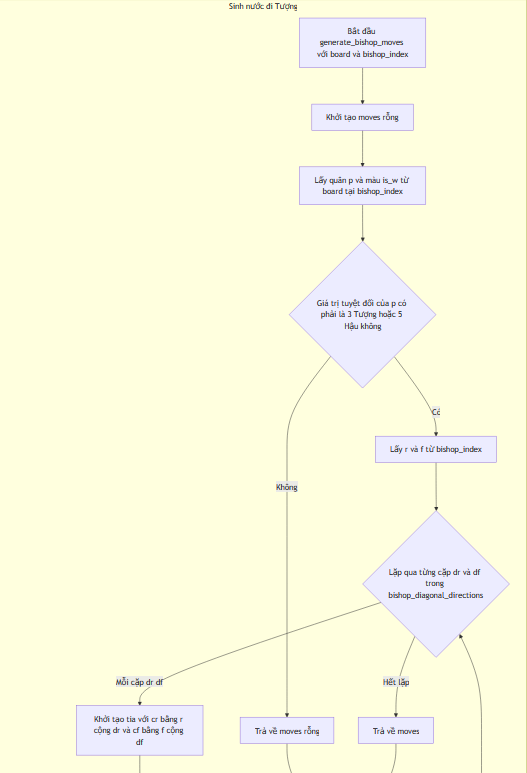
* + Quân Xe:

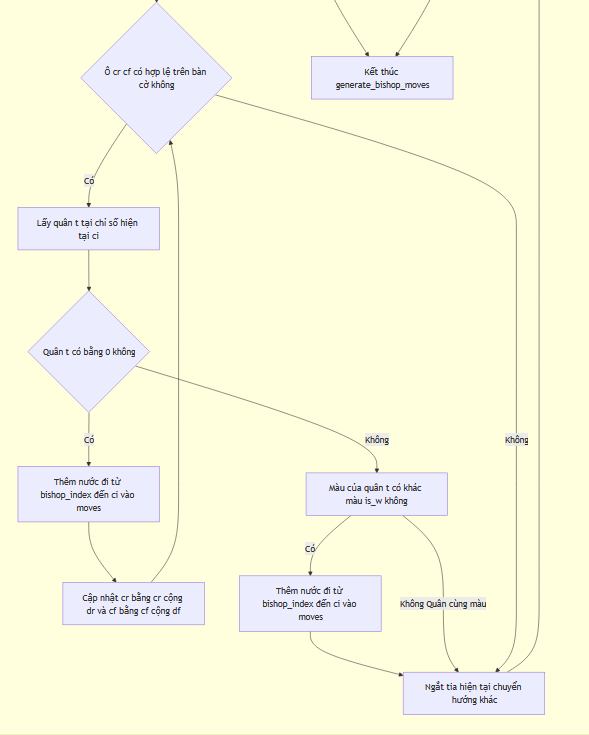




Hình 30. Sơ đồ hàm tạo nước đi cho quân Xe.

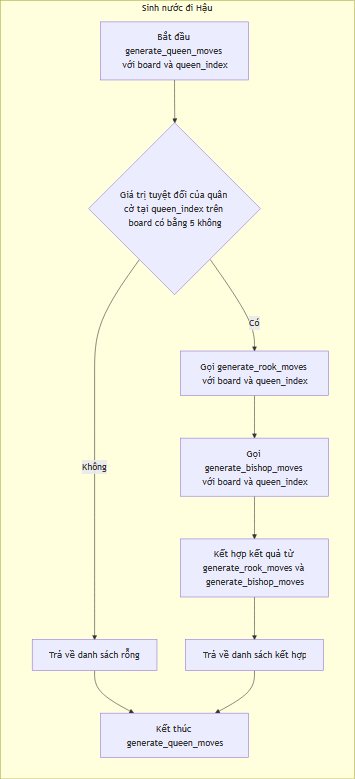
* + Quân Tượng:





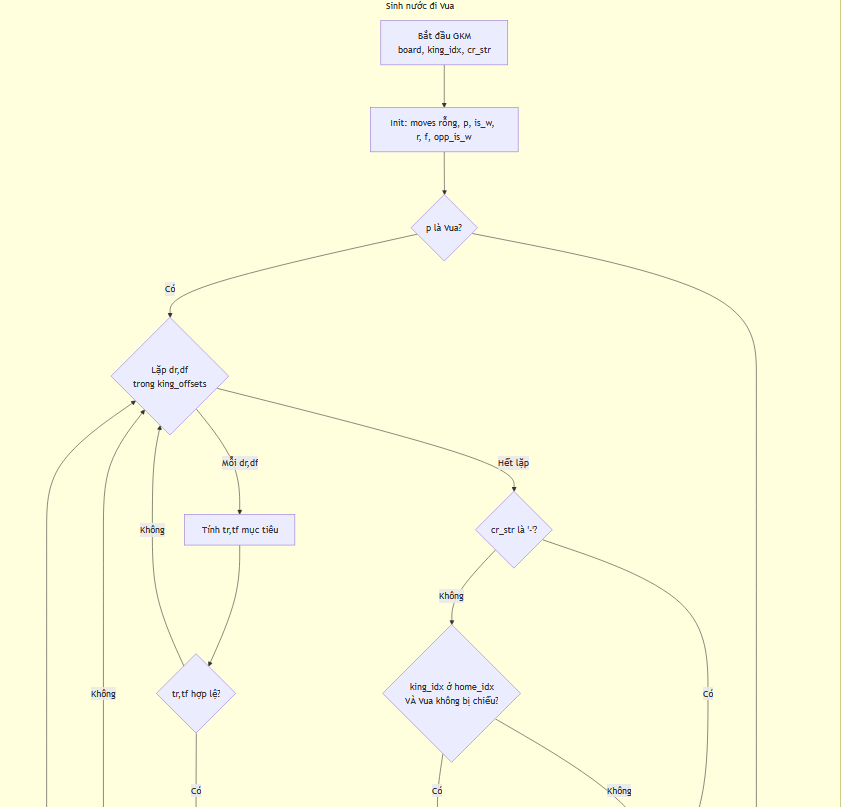
Hình 31. Sơ đồ hàm tạo nước đi cho quân Tượng

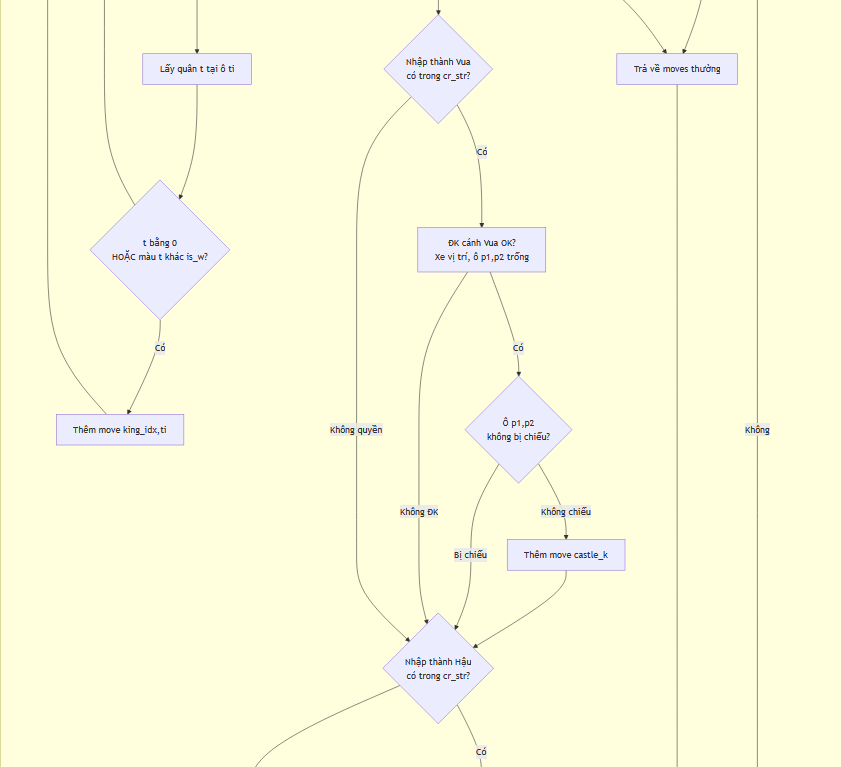
* + Quân Hậu:

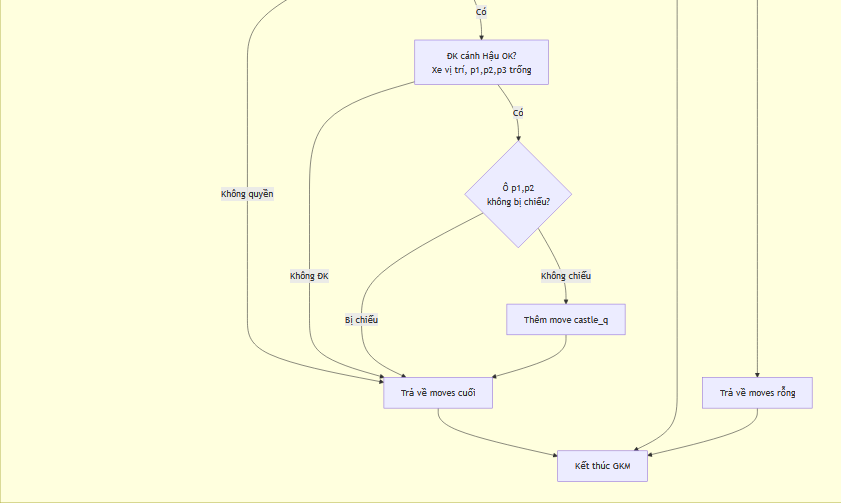


Hình 32. Sơ đồ hàm tạo nước đi cho quân Hậu.

* + Quân Vua:

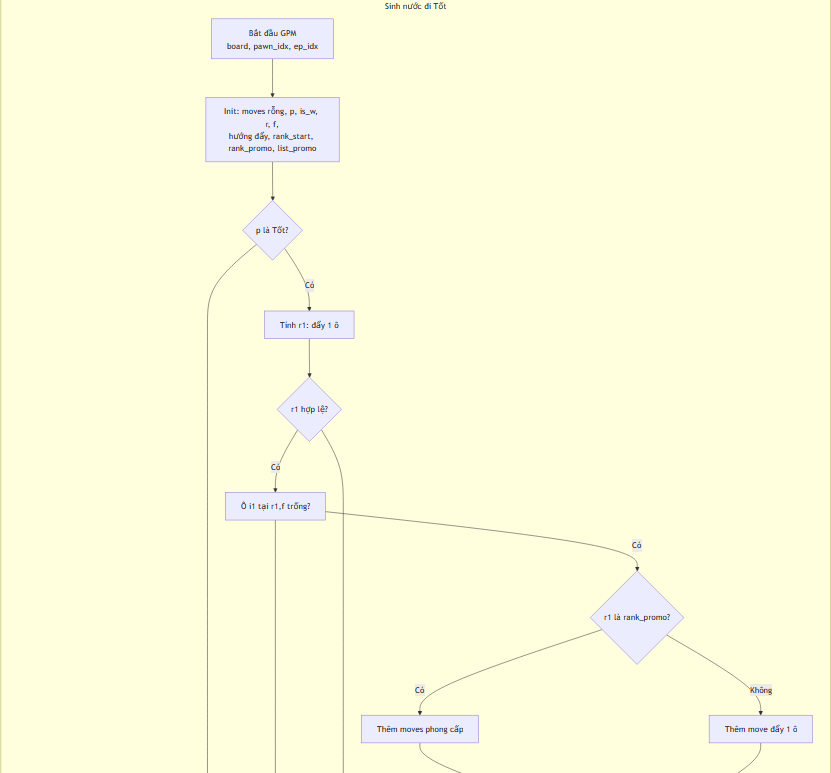


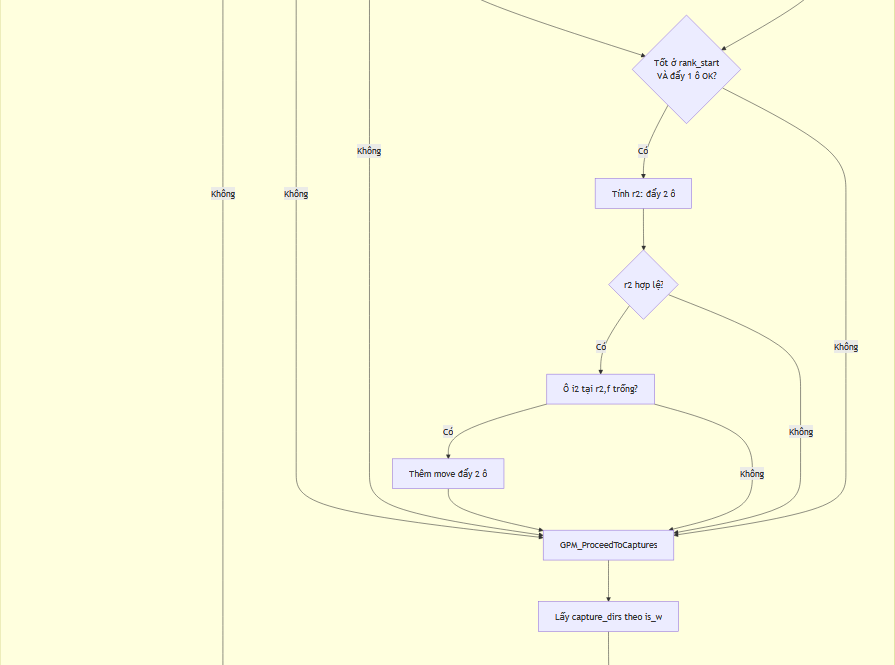


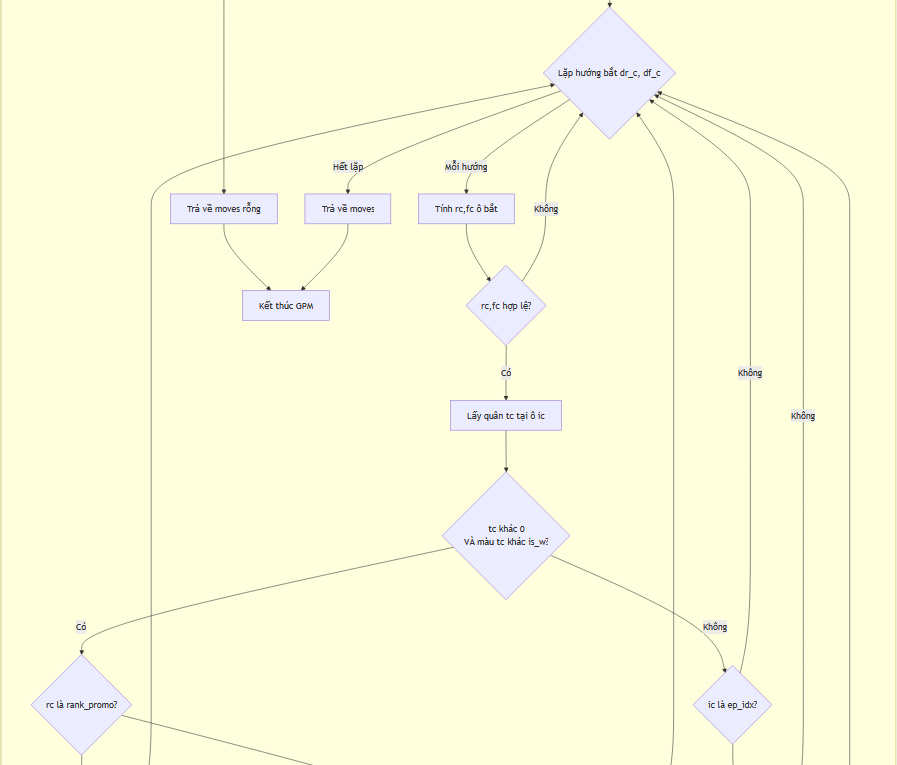


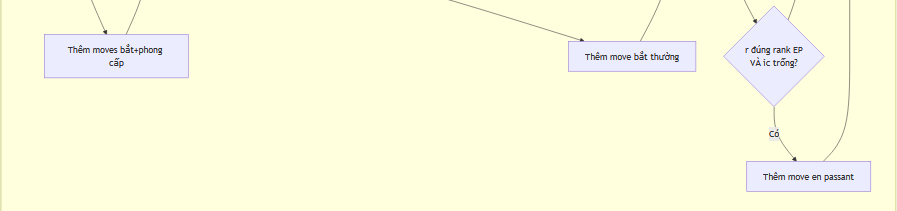
Hình 33. Sơ đồ hàm tạo nước đi cho quân Vua.

* Quân Tốt









Hình 34. Sơ đồ hàm tạo nước đi cho quân Tốt.

**CHƯƠNG 4: Cài đặt và thử nghiệm**

* 1. **PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**
     1. **Kiến trúc tổng quan**

Hệ thống AI chơi cờ vua được thiết kế dựa trên thuật toán tìm kiếm trạng thái Minimax kết hợp với cắt tỉa Alpha-Beta và bảng đánh giá vị trí quân cờ (Piece-Square Table). Kiến trúc này giúp tối ưu hóa chiến lược tìm kiếm nước đi, tăng hiệu suất tính toán và cải thiện khả năng đánh giá vị trí trên bàn cờ.

Hệ thống bao gồm các thành phần chính sau:

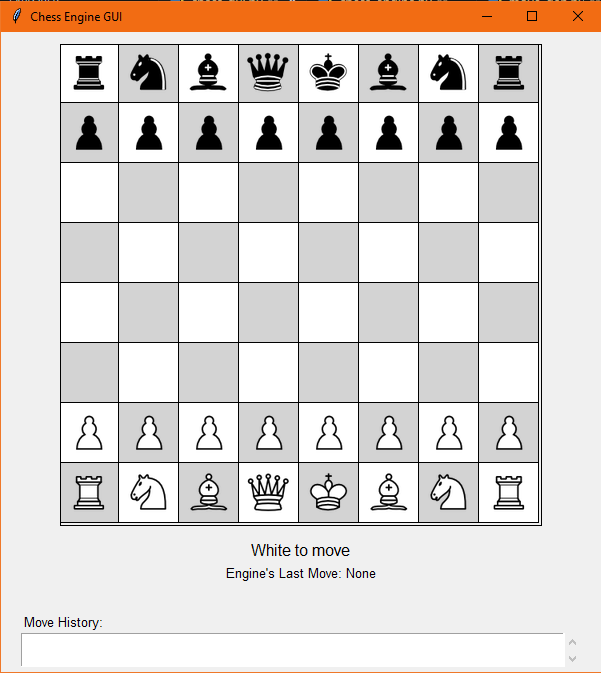
* **Mô-đun nhập liệu và hiển thị bàn cờ**: Giao diện tương tác với người dùng, cung cấp các công cụ quản lý trận đấu, ghi nhớ lịch sử nước đi và hệ thống nhận diện trạng thái bàn cờ.
* **Mô-đun tìm kiếm nước đi**: Sử dụng thuật toán Minimax với cắt tỉa Alpha-Beta để tối ưu hóa tìm kiếm nước đi tốt nhất.
* **Mô-đun đánh giá trạng thái**: Dựa trên bảng Piece-Square Table để định lượng giá trị vị trí của các quân cờ.
  + 1. **Công nghệ sử dụng**

Hệ thống được xây dựng dựa trên các công nghệ:

* Ngôn ngữ lập trình: Python
* Giao diện người dùng: Tkinter
  1. **Thiết kế giao diện**

Giao diện cần đảm bảo:

* Hiển thị bàn cờ trực quan.
* Cho phép đi cờ bằng cách nhấn vào ô hợp lệ.
* Hiển thị lịch sử các nước đã đi.



# CHƯƠNG 5: Tổng kết

Chương trình đã đạt được một số thành công đáng chú ý, nhưng vẫn còn những điểm cần cải thiện để nâng cao trải nghiệm người dùng và khả năng sử dụng của ứng dụng.

# KẾT QUẢ ĐẠT ĐƯỢC

**Phần mềm cờ vua:** Chương trình đã triển khai một bàn cờ vua gồm đầy đủ quân và đầy đủ luật sử dụng thuật toán Minimax, cắt tỉa Alpha-Beta và thuật toán suy nghiệm Piece-Square Table. Mô hình này đã hoạt động hiệu quả và cho kết quả độ chính xác và nhanh chóng.

**Giao diện người dùng:** Giao diện người dùng được thiết kế trực quan, với phần bàn cờ và phần lịch sử nước cờ.

**Đánh giá hiệu suất:** Phần mềm cờ vua đưa ra các nước đi nhanh chóng và chính xác với độ chính xác cao và có chiến thuật.

# ĐIỂM CẦN CẢI THIỆN

**Mô hình khác và dữ liệu nâng cao:** Chương trình hiện chỉ sử dụng thuật toán cổ điển thuần túy. Để làm cho ứng dụng phong phú hơn, có thể cân nhắc mở rộng để hỗ trợ mô hình học tăng cường cùng học sâu như cây tìm kiếm Monte Carlo và nhiều bộ dữ liệu opening khác nhau .

**Tính tương tác người dùng:** Tính tương tác của người dùng có thể được cải thiện bằng cách thêm tính năng đi lại, điều chỉnh độ sâu mô phỏng, và chế độ chơi mới như tính giờ, 2 người, giả lập 2 máy, v.v. Điều này sẽ tăng tính linh hoạt và sự thoải mái cho người dùng.

**Độ “thông minh” của phần mềm:** Phần mềm tuy đã đạt đến độ hoàn thiện đủ để chơi được với người nhưng khi đấu với các phần mềm máy khác có thể thấy rõ các thiếu sót trong khâu đưa ra quyết định.

# HƯỚNG PHÁT TRIỂN

**Tích hợp mô hình khác:** Nâng cao chương trình bằng cách tích hợp mô hình học máy tăng cường, từ cây quyết định đến mạng nơ-ron.

**Tính năng tinh vi:** Thêm tính năng như lựa chọn độ sâu, tối ưu hóa thuật toán, và hiển thị đồ thị tính độ chính xác để cung cấp cái nhìn sâu sắc hơn về hiệu suất phần mềm.

**Giao diện mở rộng:** Mở rộng giao diện để bao gồm nhiều chức năng và tùy chọn, giúp người dùng tương tác một cách dễ dàng hơn và tận hưởng trải nghiệm toàn diện hơn.

# TÀI LIỆU THAM KHẢO

Admin1. (2023, 05 18). *Hitechpro*. Được truy lục từ https://hitechpro.vn/blogs/news/cong-nghiep-4-0-va-tri-tue-nhan-tao-ai-huong-toi-mot-tuong-lai-ky-d

Bình, N. V. (2023). *Nghiên cứu các giải pháp định vị trong nhà hiệu quả dựa trên dữ liệu sóng không dây*. ( Học viện Khoa học và Công nghệ, Viện Hàn lâm Khoa học và Công nghệ Việt Nam) Được truy lục từ https://luanvan.moet.gov.vn/?page=1.3&view=42528

*Datagy*. (2022, 01 05). Được truy lục từ https://datagy.io/python-sklearn-linear-regression/

David Silver, T. H. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 1140-1144.

Donald E. Knuth, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 293-326.

FIDE. (2022, December 31). *International Chess Federation's Handbook*. Được truy lục từ International Chess Federation's Website: https://handbook.fide.com/chapter/E012018

hemavatisabu. (2022, 02 03). *GeeksForGeeks*. Được truy lục từ https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/

Hoà, L. Q. (2019, 09 27). *123doc*. Được truy lục từ https://123docz.net/document/5690209-ung-dung-thuat-toan-k-lang-gieng-gan-nhat-trong-phan-loai-van-ban-tin-tuc-theo-chu-de.htm

Kumar, B. (2021, 12 16). *PytthonGuides*. Được truy lục từ https://pythonguides.com/scikit-learn-decision-tree/

Lewis, A. (2020). Magnus Carlsen breaks record for longest unbeaten run in chess. *CNN*.

Narula, M. (2023, 10 10). *DelftStack*. Được truy lục từ https://www.delftstack.com/howto/python/lasso-regression-in-python/

PGS. TS. Phạm Văn Cường, M. T. (2023). *Phát triển game cờ vua tự động bằng phương pháp học sâu.* Hà Nội: Học viện Công nghệ Bưu chính Viễn thông.

PythonMaster. (2023, 02 18). *PythonBaba*. Được truy lục từ https://pythonbaba.com/how-to-group-by-multiple-columns-in-a-pandas-dataframe/#

Schröder, E. (2002). *How REBEL Plays Chess*. Được truy lục từ http://members.home.nl/matador/chess840.htm.

*Scikit Learn*. (không ngày tháng). Được truy lục từ https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

*Scikit Learn*. (không ngày tháng). Được truy lục từ https://scikit-learn.org/stable/supervised\_learning.html

Singh, D. (2019, 05 17). *PluralsSght*. Được truy lục từ https://www.pluralsight.com/guides/linear-lasso-ridge-regression-scikit-learn

Trang, P. T. (2021, 06 08). *123doc*. (TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG) Được truy lục từ https://123docz.net/document/8576011-he-mien-dich-nhan-tao-va-ung-dung-cho-loc-thu-rac.htm

Tuấn, N. N. (2020). Được truy lục từ https://ptithcm.edu.vn/wp-content/uploads/2023/07/2020\_HTTT\_LeHoangBao\_LV.pdf

*Vector Linux*. (2023, 11 07). Được truy lục từ https://vectorlinux.com/es-get-all-records-python/

*Wenku*. (2023, 09 13). Được truy lục từ https://wenku.csdn.net/answer/08e52f82af494f599861c8c37860b064