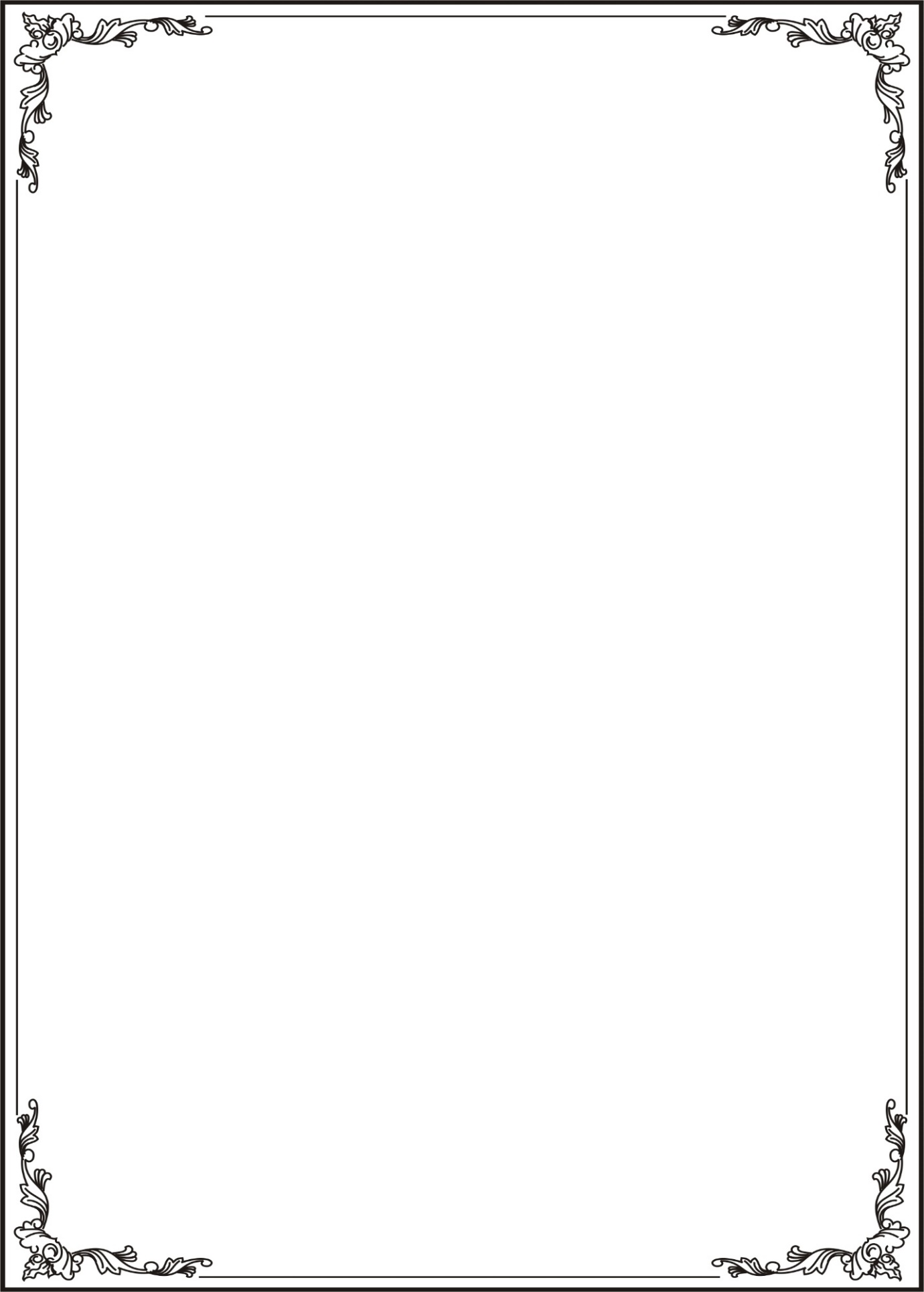
**ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**

****

**BÁO CÁO ĐỒ ÁN CUỐI KỲ**

**MÔN: TRÍ TUỆ NHÂN TẠO**

**Đề tài: Xây dựng game cờ vua**

**Giảng viên hướng dẫn**: **ThS. Huỳnh Thị Thanh Thương**

**Lớp: CS106.H21.KHTN**

**Sinh viên thực hiện:**

* Lê Trịnh Khánh Duy 15520159
* Thái Quang Minh 15520494

***TP. Hồ Chí Minh, ngày 21 tháng 6 năm 2017***

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**MỤC LỤC**

[LỜI CẢM ƠN 4](#_Toc487058435)

[NHẬN XÉT CỦA GIẢNG VIÊN 6](#_Toc487058436)

[CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN 7](#_Toc487058437)

[1.1. Nguồn gốc trò chơi 7](#_Toc487058438)

[1.2. Mô tả trò chơi 8](#_Toc487058439)

[1.3. Trí tuệ nhân tạo đối với trò chơi 9](#_Toc487058440)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 10](#_Toc487058441)

[2.1. Đặt vấn đề 10](#_Toc487058442)

[2.2. Giải thuật Minimax 10](#_Toc487058443)

[2.2.1. Giới thiệu 10](#_Toc487058444)

[2.2.2. Ý tưởng 11](#_Toc487058445)

[2.2.3. Mã giả 12](#_Toc487058446)

[2.2.4. Ví dụ (trò chơi TIC-TAC-TOE) 13](#_Toc487058447)

[2.3. Giải thuật Alphabeta 14](#_Toc487058448)

[2.3.1. Giới thiệu 14](#_Toc487058449)

[2.3.2. Ý tưởng 14](#_Toc487058450)

[2.3.3. Mã giả 15](#_Toc487058451)

[2.3.4. Ví dụ cụ thể 16](#_Toc487058452)

[CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ 17](#_Toc487058453)

[3.1. Phân tích bài toán 17](#_Toc487058454)

[3.1.1. Tính phức tạp của bài toán 17](#_Toc487058455)

[3.1.2. Ý tưởng giải quyết bài toán 18](#_Toc487058456)

[3.1.2.1. Biểu diễn bàn cờ và các bước di chuyển 18](#_Toc487058457)

[3.1.2.2. Hàm đánh giá 20](#_Toc487058458)

[3.1.2.3. Tìm kiếm 21](#_Toc487058459)

[3.2. Cấu trúc dữ liệu và cách biểu diễn các trạng thái của bài toán 22](#_Toc487058460)

[3.2.1. Cách biễu diễn trạng thái của bài toán 22](#_Toc487058461)

[3.2.2. Cấu trúc dữ liệu của bài toán 24](#_Toc487058462)

[3.2.2.1. Tìm kiếm nước đi tốt nhất 24](#_Toc487058463)

[3.2.2.2. Biểu diễn bàn cờ và các thực hiện nước đi 27](#_Toc487058464)

[3.2.2.3. Biểu diễn các nước đi có thể đi 30](#_Toc487058465)

[3.2.2.4. Đánh giá 33](#_Toc487058466)

[3.3. Các vấn đề của thuật giải 36](#_Toc487058467)

[3.3.1. Xây dựng mô hình Alpha-Beta để tìm kiếm 36](#_Toc487058468)

[3.3.2. Xây dựng các nước di chuyển có thể đi của quân cờ 47](#_Toc487058469)

[3.3.2.1. Quân xe *(Loại I)* 48](#_Toc487058470)

[3.3.2.2. Quân tượng *(Loại I)* 52](#_Toc487058471)

[3.3.2.3. Quân Mã (Loại I) 56](#_Toc487058472)

[3.3.2.4. Quân Hậu (Loại I): 59](#_Toc487058473)

[3.3.2.5. Quân Tốt (Loại I, Loại II) 64](#_Toc487058474)

[3.3.2.6. Quân Vua (Loại I và Loại III) 73](#_Toc487058475)

[3.3.2.7. Xây dựng, lấy chuỗi nước đi cho quân cờ loại I. 81](#_Toc487058476)

[3.3.3. Xây dựng cơ chế kiểm tra sự an toàn (bị chiếu) của quân Vua 83](#_Toc487058477)

[3.3.4. Xây dựng mô hình đánh giá không gian trạng thái 88](#_Toc487058478)

[3.3.4.1. Đánh giá dựa trên số quân cờ hiện hữu (Material Balance). 93](#_Toc487058479)

[3.3.4.2. Đánh giá dựa trên từng vị trí của bàn cờ (Piece-square value). 96](#_Toc487058480)

[3.3.4.3. Đánh giá dựa trên tính cơ động quân cờ (Mobility Score) 102](#_Toc487058481)

[3.3.4.4. Đánh giá dựa trên độ tấn công của đối thủ (Attack Score) 104](#_Toc487058482)

[3.3.5. Xây dựng cơ chế thay đổi góc nhìn 106](#_Toc487058483)

[3.3.6. Xây dựng cơ chế chuyển và phục hồi trạng thái 108](#_Toc487058484)

[3.3.6.1. Thực hiện nước đi (ApplyMove): 110](#_Toc487058485)

[3.3.6.2. Phục hồi nước đi (UndoMove): 114](#_Toc487058486)

[CHƯƠNG 4: ỨNG DỤNG 117](#_Toc487058487)

[4.1. Giới thiệu chương trình ứng dụng 117](#_Toc487058488)

[4.1.1. Yêu cầu và chức năng của chương trình 117](#_Toc487058489)

[4.1.2. Thiết kế giao diện chương trình 117](#_Toc487058490)

[4.1.2.1. Sơ đồ liên kết các màn hình 117](#_Toc487058491)

[4.1.2.2. Đặc tả giao diện 117](#_Toc487058492)

[4.1.2.3. Mô tả chi tiết các màn hình 118](#_Toc487058493)

[4.1.3. Hướng dẫn sử dụng chương trình 123](#_Toc487058494)

[4.2. Cài đặt 125](#_Toc487058495)

[4.2.1. Ngôn ngữ và công cụ sử dụng 125](#_Toc487058496)

[4.2.2. Các hàm và đoạn code chính của chương trình 127](#_Toc487058497)

[CHƯƠNG 5: KẾT LUẬN 129](#_Toc487058498)

[5.1. Kết quả đạt được 129](#_Toc487058499)

[5.2. Hạn chế 129](#_Toc487058500)

[5.3. Hướng phát triển 129](#_Toc487058501)

[TÀI LIỆU THAM KHẢO 130](#_Toc487058502)

[PHỤ LỤC 133](#_Toc487058503)

# LỜI CẢM ƠN

Đầu tiên, nhóm xin chân thành gửi lời cảm ơn đến tập thể quý Thầy, Cô Trường Đại Học Công nghệ Thông tin – Đại Học Quốc Gia Thành phố Hồ Chí Minh.

Đặc biệt, nhóm xin gửi lời cảm ơn và lòng biết ơn sâu sắc với cô Huỳnh Thị Thanh Thương – giảng viên môn Trí tuệ nhân tạo. Cô đã trực tiếp hướng dẫn, giải đáp các thắc mắc của nhóm trong quá trình thực hiện đồ án.

Trong thời gian 2 tháng thực hiện đề tài, nhóm đã vận dụng những kiến thức nền tảng đã được tích lũy và phối hợp với việc tự học hỏi, nghiên cứu những kiến thức mới. Từ đó, nhóm đã hoàn thành được đồ án. Trong quá trình thực hiện, kiến thức của nhóm còn hạn chế nên không thể tránh khỏi những thiếu sót. Chính vì vậy, nhóm mong được những ý kiến đóng góp từ phía Giảng viên, nhẳm hoàn thiện những thiếu sót, qua đó hoàn thiện hành trang của nhóm trong tương lai.

Xin chân thành cảm ơn!

*Lê Trịnh Khánh Duy – Thái Quang Minh*

# NHẬN XÉT CỦA GIẢNG VIÊN

**………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………**

# CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN

## Nguồn gốc trò chơi

Khi nhắc đến lịch sử của cờ vua [[1]](https://en.wikipedia.org/wiki/Chess#History), có rất nhiều quốc gia cho rằng chính mình là nơi sản sinh ra môn thể thao trí tuệ bậc nhất này. Thế nhưng, khi các nhà sử học tìm kiếm và nghiên cứu, thì phát hiện môn cờ vua là một trò chơi giải trí xuất hiện vào khoảng thế kỷ thứ 6 ở Ấn Độ trong thời kỳ đế chế Gupta, lúc này nó có tên gọi là Chaturanga*.* Lúc này trò chơi gồm nhiều loại binh chủng xuất hiện trên một cờ, bao gồm : bộ binh, kỵ binh, tượng binh và xa binh tương ứng các quân cờ hiện đại hiện nay là chốt, mã, tượng và xe. Cũng trong những năm 600 sau công nguyên, ở đất nước Ba Tư, tên trò chơi được gọi là Chatrang, nhưng những luật chơi đã bắt đầu phát triển cao hơn. Đặc biệt 2 thuật ngữ mới ra đời là: “Shāh!” khi chiếu vào Vua đối phương và “Shāh māt!” khi chiếu bí Vua khiến ván cờ kết thúc.

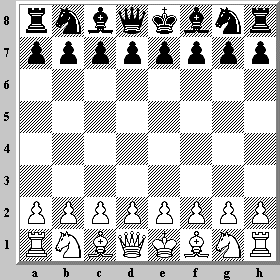
Nhờ những cuộc chinh phạt của đế quốc Ba Tư, mà trò chơi này đã được đưa đến, đẩy mạnh đến các quốc gia, tùy vào ngữ điệu của địa phương mà tên trò chơi được thay đổi như: Shatraj (ở các quốc gia Tây Á), Shatarej (ở Maroc), Scacchi (ở Italia), Schack (ở Thụy Điển),….

Cờ vua đã du nhập đến nhiều vùng đất khác nhau trên thế giới, cũng từ đó mà số lượng người chơi cờ vua càng ngày càng đông hơn. Ban đầu, ở các cùng đất khác nhau, sẽ có những luật khác đôi chút với những nơi còn lại. Thế nhưng khi số lượng người chơi đã đủ đông, thì nảy sinh ra phải có một quy định thống nhất về luật chơi. Trò chơi mà mọi người dù khác biệt về văn hóa, ngôn ngữ đều phải chơi được, khoảng cuối thể kỷ 15 của thời Phục Hưng thì luật chơi được phát triển mạnh mẻ và sớm thành một hế thống.

Dần dần qua hàng trăm năm, khi mà cờ vua dần hoàn thiện và đến mọi người, luật chơi cũng từ đó mà hợp lý, hoàn thiện. Những câu lạc bộ tổ chức, trung tâm cờ xuất hiện, đó là nơi mọi người yêu cờ học hỏi lẫn nhau. FIDE – Liên đoàn cờ vua Quốc tế cũng từ đó mà hình thành, nhằm mục tiêu xây dựng và phát triển hệ thống thi đấu Quốc gia và Quốc tế.

## Mô tả trò chơi

Cờ vua [[2]](https://en.wikipedia.org/wiki/Chess) là một môn thể thao trong những trò chơi trí tuệ phổ biến nhất trên thế giới, thu hút hơn 600 triệu người chơi [[3]](http://en.chessbase.com/post/che-redux-how-many-people-play-che-) trên thế giới mỗi ngày. Cờ vua không chỉ đơn thuần là một trò chơi, mà còn được biết đến chính là nơi tập hợp của nghệ thuật, khoa học, và thể thao.

Cờ vua là một trò chơi trên bàn và là môn thể thao đối kháng trí tuệ 2 người. Trò chơi diễn ra trên một bảng hình vuông, gọi là bàn cờ. Trên bàn cờ bao gồm 8 hàng (được đánh số từ 1 đến 8), 8 cột (được đánh các chữ cái từ a đến h), tạo ra 64 ô vuông với các màu đậm và nhạt xen kẽ nhau, ô màu đậm ở cuối hàng bên trái của mỗi người chơi. Trò chơi bao gồm các quân cờ màu trắng và đen, người nào cầm quân trắng luôn là người đi đầu, người kia cầm quân đen thì đi sau. Mỗi người sẽ thực hiện nước đi của mình, sau khi đối phương đã đi xong một nước cờ. Các quân cờ của mỗi bên bao gồm:

* 8 quân Tốt.
* 2 quân Xe.
* 2 quân Mã.
* 2 quân Tượng.
* 1 quân Hậu.
* 1 quân Vua.

Mỗi quân cờ có nước đi và nhiệm vụ khác nhau [[4]](https://en.wikipedia.org/wiki/Chess#Movement), nhưng tất cả chúng đều có điểm chung là phải bảo vệ quân Vua. Cờ vua không chỉ là nơi biểu diễn sức mạnh, uy lực tấn công, mà còn là nơi phô diễn nghệ thuật phòng thủ. Người chơi biết phối hợp hài hòa giữa nhu và cương tạo ra một chiến thuật, thế trận liên hoàn và là linh hồn của ván cờ, dẫn người chơi đến chiến thắng.

## Trí tuệ nhân tạo đối với trò chơi

Sự kiện chính và nổi bật ắt hẳn là sự kiện ngày 11 tháng 5 năm 1997, hơn 20 năm, ngày mà siêu máy tính Deep Blue [[5]](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)) của IBM đã đánh bại đại kiện tướng cờ vua người Nga, Gary Kimovich Kasparov. IBM đã tung ra phiên bản phần mềm mới đã được cải tiến nhiều cùng với hệ siêu phần cứng gồm nhiều bộ vi xử lý chạy song song. Deep Blue được thiết kế chuyên để "đấu với Kasparov", tất cả những nước đi khai cuộc mà Kasaprov đã từng sử dụng đều được cài trong cho nó. Và lần này IBM đã thành công, trong trận đấu 6 ván Deep Blue đã hạ Kasparov với tỷ số 3,5-2,5. Tại ván đấu cuối cùng, Deep Blue phát hiện sai lầm của Kasparov từ nước đi khai cuộc và quyết định thí quân phá vỡ thế trận của Kasparov ngay từ đầu ván cờ khiến chung cuộc Kasparov phải đầu hàng sớm. Xét về khả năng tính toán nước đi, thì không người nào có thể so sánh với máy tính cả. Trong khi siêu máy tính Deep Blue đạt khả năng tính toán 200 triệu nước mỗi giây, thì đại kiến tướng Kasparov chỉ đạt được 5 nước đi mỗi giây là cao. Tuy nhiên ông đã thi đấu ngang ngửa với siêu máy tính theo cách “kỹ năng của riêng con người”.

Thành công của Deep Blue cũng là khởi đầu của thời đại máy tính thống trị cờ vua, con người khó có khả năng thi đấu cùng với máy tính. Quả thật cờ vua – môn thể thao tạo nên rất nhiều bài toán cho các lập trình viên.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## Đặt vấn đề

Trò chơi - thường gắn liền với trí tuệ của con người được phát triển qua các nền văn minh nhân loại. Những trò chơi được phân loại theo nhiều yêu cầu như kỹ năng hay sự may rủi hoặc kết hợp cả hai yếu tố. Những trò chơi may rủi thường thấy như: tài xỉu, xì dách, bầu cua tôm cá,… chúng thường không yêu cầu tính chiến thuật. Trái lại, với những trò chơi khác yêu cầu kỹ năng như: bóng đá, cờ vua, cờ tướng, cờ vây,.. Ở những trò chơi này tồn tại song song các đối tượng, đối kháng với nhau, người chơi luôn tìm cách chiến thắng, còn đối phương luôn cản trở gây khó khăn và không bao giờ cho chúng ta đến chiến thắng. Với các nhà khoa học nghiên cứu trong lĩnh vực AI, bản chất trừu tượng của trò chơi khiến chúng trở thành một chủ đề hấp dẫn để nghiên cứu. Ví dụ, cờ vua là một trò chơi trí tuệ mang đầy chiến thuật và mang tính đối kháng cao:

* Hai người chơi sẽ thay phiên nhau đưa ra các nước đi, tuân theo các luật của cờ vua một cách chính xác. Quân vua của người nào bị chiếu hết thì người đó sẽ thua.
* Cả hai người chơi đều bình đằng, nghĩa là biết thông tin đầy đủ về các tình thế trong trò chơi. (Sự hơn kém trình độ, kinh nghiệm được thể hiện trong việc ra nước đi, chiến thuật sử dụng của mỗi người chơi).
* Nhiệm vụ của người chơi là tìm ra trong số rất nhiều nước đi hợp lệ một nước đi tốt nhất, sao cho qua một dãy các nước đi, người chơi giành được chiến thắng.

## Giải thuật Minimax

### Giới thiệu

Mỗi người chơi sẽ cố gắng tối đa hóa lợi thế của mình. Ngược lại, đối thủ của họ sẽ cố gắng hạn chế điều đó. Lượt chơi của mỗi người thay phiên nhau, vì thế để đạt được lợi thế cho mình, người chơi phải nhìn trước được các nước đi của mình lẫn đối thủ. Giải thuật Minimax cơ bản là để giảm thiểu tối đa lợi thế của đối phương và phát huy tối đa lợi thế của mình.

Là giải thuật cơ bản, được áp dụng rộng rãi trong những trò chơi mang tính đối kháng, mà điều kiện thiết yếu là dựa trên 2 giả thiết sau:

* Cả 2 người chơi đều có cùng kiến thức như nhau về không gian trạng thái của trò chơi. (Ngược lại, ở những trò chơi mà người chơi không cùng kiến thức về không gian trạng thái như trò chơi poker, cả người chia bài và các người chơi đều không biết lá bài trong tay đối thủ là quân bài gì).
* Cả 2 đối thủ đều có cùng mức cố gắng thắng như nhau.

### Ý tưởng

Trong một trò chơi gồm có 2 đối thủ, mỗi bên sẽ có nhiệm vụ tìm chiến thắng cho bản thân và làm cho đối phương thua cuộc, hoặc ít nhất là 2 bên cùng hòa. Giả sử ta đặt cho người chơi có tên là MAX và đối thủ của họ có tên là MIN:

* MAX: là trạng thái biễu diễn cho mục đích của người chơi là làm tối đa lợi thế của mình.
* MIN: là trạng thái biễu diễn cho mục đích của người chơi là hạn chế tối đa lợi thế của đối thủ.

Một trò chơi có thể định nghĩa dưới dạng *cây tìm kiếm* gồm các thành phần sau:

* Trạng thái: tình thế của trò chơi. VD: sự bố trí các quân cờ.
* Trạng thái ban đầu: tình thế của trò chơi lúc ban đầu.
* Các toán tử: là các luật quy định của những nước đi hợp lệ.
* Các trạng thái kết thúc: các tình thế khi trò chơi kết thúc.
* Hàm kết cuộc: trả về một giá trị tương ứng với mỗi trạng thái kết thúc.

*Cây tìm kiếm* sẽ phân thành các lớp MAX và MIN. Bởi vì không gian trạng thái của cờ vua rất lớn vì thế việc giới hạn độ sâu của không gian tìm kiếm là cần thiết.

Với 1 Node (trạng thái) bất kỳ:

* Nếu là Node lá thì: trả về giá trị được tính toán bởi hàm kết cuộc. Ví dụ: 1 là MAX thắng, 0 là hòa, -1 là MIN thắng
* Các Node thuộc lớp MAX -> gán giá trị lớn nhất trong tất cả các node con của nó
* Các Node thuộc lớp MIN -> gán giá trị nhỏ nhất trong tất cả các node con của nó

### Mã giả

Mã giả với độ sâu giới hạn của giải thuật Minimax  [[6]](https://en.wikipedia.org/wiki/Minimax)*:*

***function*** *minimax(node, depth, maximizingPlayer)*

***if*** *depth = 0 or node is a terminal node*

*return the heuristic value of node*

***if*** *maximizingPlayer*

*bestValue := −∞*

***for*** *each child of node*

*v := minimax(child, depth − 1, FALSE)*

*bestValue := max(bestValue, v)*

*return bestValue*

*else (\* minimizing player \*)*

*bestValue := +∞*

*for each child of node*

*v := minimax(child, depth − 1, TRUE)*

*bestValue := min(bestValue, v)*

*return bestValue*

*for each child of node*

*v := minimax(child, depth − 1, TRUE)*

*bestValue := min(bestValue, v)*

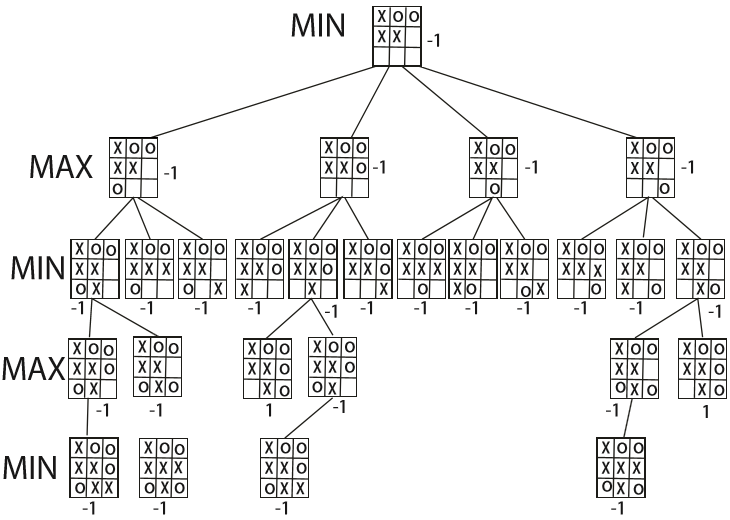
*return bestValue*

### Ví dụ (trò chơi TIC-TAC-TOE)

Là trò chơi gồm hình vuông 9 ô (3x3) và 2 người chơi X và O. Mỗi người chơi sẽ lần lượt đánh dấu của mình vào các ô. Ai có đủ 3 dấu X (hoặc O) theo hàng dọc, ngang, chéo thì người đó sẽ chiến thắng.

Giả sử MAX là quân O, MIN là quân X.

Ở các Node lá có giá trị: 1 nếu O thắng, 0 nếu hòa, -1 nếu X thắng



=> Ở tình thế hiện tại thì mọi nước đi có thể của O luôn có cùng một giá trị là -1, nghĩa là dù O có đi như thế nào thì X cũng sẽ chiến thắng.

1. **Nhận xét**

**Ưu điểm**

Giải thuật minimax mang tinh chất vét cạn vì sẽ tìm kiếm tất cả mọi nước đi có thể của người chơi và đối thủ để tìm ra nước đi tốt nhất.

**Khuyết điểm**

Đối với các trò chơi có không gian trạng thái lớn như caro, cờ tướng… việc chỉ áp dụng giải thuật Minimax có lẽ không còn hiệu quả nữa do sự bùng nổ tổ hợp quá lớn. Giải thuật áp dụng nguyên lý vét cạn không tận dụng được thông tin của trạng thái hiện tại để lựa chọn nước đi, vì duyệt hết các trạng thái nên tốn thời gian. [[7]](https://www.stdio.vn/articles/read/283/giai-thuat-tim-kiem-minimax)

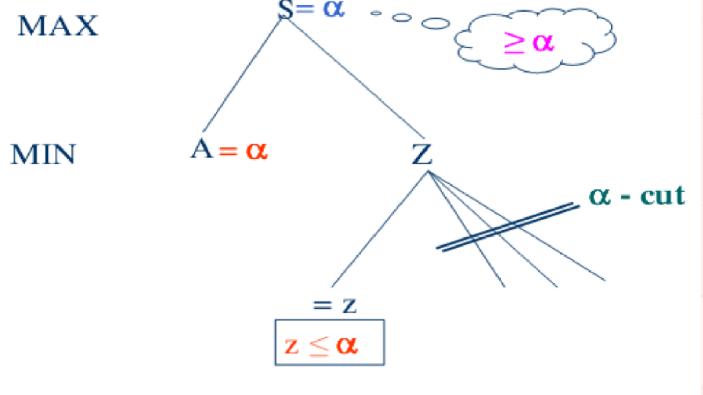
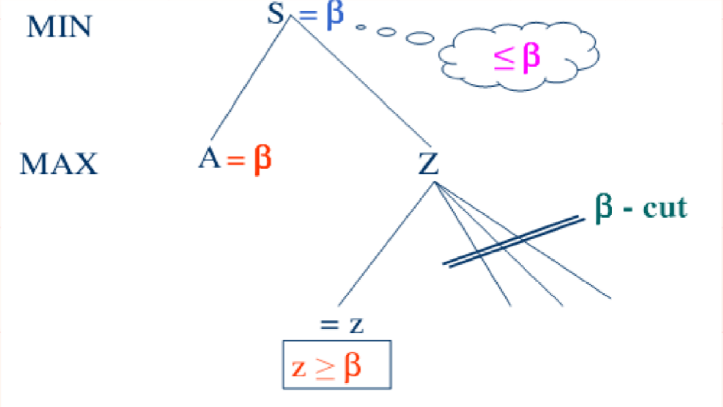
## Giải thuật Alphabeta

### Giới thiệu

Giải thuật cắt tỉa Alpha-beta được nhiều nhà khoa học máy tính đề xuất ý tưởng và không ngừng được cải tiến cho đến ngày nay. Giải thuật này là giải thuật tang cường của giải thuật tìm kiếm Minimax nhằm hỗ trợ và giảm bớt các không gian trạng thái trong cây trò chơi, giúp thuật toán Minimax có thể tìm kiếm sâu và nhanh hơn. Giải thuật cắt tỉa Alpha-beta có nguyên tắc đơn giản "Nếu biết là trường hợp xấu thì không cần phải xét thêm". [[8]](https://www.stdio.vn/articles/read/564/giai-thuat-cat-tia-alpha-beta)

### Ý tưởng

Giải thuật Alphabeta là giải thuật cải tiến từ giải thuật minimax, sẽ cắt bỏ những node không cần thiết trên cây không gian tìm kiếm với mục đích giảm bớt số đỉnh cần đánh giá. Giải thuật sử dụng 2 giá trị alpha và beta. Gán và . Các Node MAX có một giá trị alpha, Các Node MIN có một giá trị beta .Khi chưa có alpha và beta xác định thì thực hiện tìm kiếm sâu (depth-first) để xác định được alpha, beta, và truyền ngược lên các nút cha. Khi alpha >= beta thì sẽ tiến hành cắt tỉa. VD:



### Mã giả

***function*** *alphabeta(node, depth, α, β, maximizingPlayer)* [**[9]**](https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning)

***if*** *depth = 0 or node is a terminal node*

***return*** *the heuristic value of node*

***if*** *maximizingPlayer*

*v := -∞*

***for*** *each child of node*

*v := max(v, alphabeta(child, depth – 1, α, β, FALSE))*

*α := max(α, v)*

***if*** *β ≤ α*

*break (\* β cut-off \*)*

***return*** *v*

***else***

*v := +∞*

***for*** *each child of node*

*v := min(v, alphabeta(child, depth – 1, α, β, TRUE))*

*β := min(β, v)*

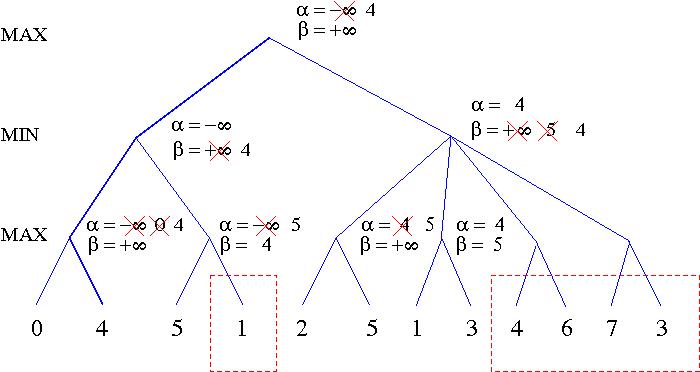
*if β ≤ α*

*break (\* α cut-off \*)*

***return*** *v*

### Ví dụ cụ thể

[[10]](http://www.cse.unsw.edu.au/~cs9414/15s1/tut/sol/wk04sol.html)



# CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ



## Phân tích bài toán

### Tính phức tạp của bài toán

Để giải quyết được bài toán về trí trí tuệ nhân tạo trong cờ vua, ta cần phải giải quyết được vấn đề thời gian thực thi và biểu diễn không gian trạng thái. Giải quyết bài toán cờ vua là phải tìm ra một chuỗi các phương pháp có kết nối với nhau, từ đó tìm kiếm được bước di chuyển tốt nhất trong tập hợp chuỗi các nước cờ để dẫn đến chiến thắng.

|  |  |  |  |
| --- | --- | --- | --- |
| **Độ sâu** | **Các nốt đếm được** | **Độ sâu** | **Các nốt đếm được** |
| 2 | 900 | 6 | 729,000,000 |
| 3 | 27,000 | 7 | 21,870,000,000 |
| 4 | 810,000 | …….. | …… |
| 5 | 24,300,000 | 40 |  |

Để tìm kiếm một kỹ thuật cho việc tìm kiếm không gian trạng thái trong bài toán cờ vua, ta cùng điểm qua những con số phức tạp xảy ra trong bài toán. Theo nghiên cứu tính toán của Shannon [[11]](https://en.wikipedia.org/wiki/Claude_Shannon), trung bình cứ một nước cờ có thể sản xinh ra 30 nước cờ khác [[12]](http://www.ee.ufpe.br/codec/Programming_a_computer_for_playing_chess.shannon.062303002.pdf). Từ đây, muốn tạo ra một dữ liệu hoàn chỉnh bao gồm tất cả các nước đi có sinh ra, ta có thể gọi đệ quy cùng với các quy tắc của trò chơi. Bằng cách này, cây trò chơi được tạo ra, nốt gốc là nơi bắt đầu của ván cờ, với quân trắng là quân đi trước, ta tạo ra được 20 nốt con là những bước di chuyển của quân cờ (bao gồm 16 nước đi của quân tốt, và 4 nước đi của quân mã). Tất cả các nốt con tạo ra được xem là không gian trạng thái tìm kiếm của trò chơi ở 1 nước đi tiếp theo (độ sâu 1). Các nốt con trạng thái vừa tìm được chỉ là bước khởi đầu đơn giản, hãy thử tính toán xem với nhiều nước đi hơn thì không gian trạng thái sẽ như thế nào. Cũng như Shannon, ông đã tính toán ra được vào một ván cờ trung bình kết thúc vào khoảng 40 nước đi.

Với số lượng không gian trạng thái vô cùng lớn (vượt qua số lượng nguyên tử của vũ trụ ), một máy tính phải mất đến năm để tính toán hết được 40 nước đi để chọn ra nước đi đầu tiên.

### Ý tưởng giải quyết bài toán

Để giải quyết bài toán với số lượng trạng thái phát sinh lớn như vậy, ta cần giải quyết 3 công việc chính sau:

* Biểu diễn bàn cờ và các bước di chuyển
* Hàm đánh giá
* Tìm kiếm nước đi tốt nhất

#### Biểu diễn bàn cờ và các bước di chuyển

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| r | n | b | q | k | b | n | r |
| p | p | p | p | p | p | p | p |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| P | P | P | P | P | P | P | P |
| R | N | B | Q | K | B | N | R |
| *Ví dụ: Trạng thái của bàn cờ ban đầu theo luật cờ vua.* | | | | | | | |

Để biểu diễn trạng thái bàn cờ, ta sử dụng một mảng 2 chiều String[8][8] để lưu vị trí của tất cả các quân cờ. Trong đó bàn cờ khởi tạo là bàn cờ chứa vị trí của các quân cờ bắt đầu. Chúng ta cần phải xác định được vị trí của các quân cờ ở đâu trên bàn cờ. Ta thường gọi các quân cờ lần lượt như sau: r, n, b, q, k, p là các quân cờ đen Rook (Xe), Knight (Ngựa), Bishop (Tượng), Queen (Hậu), King (Vua), Pawn (Tốt) và R, N, B, Q, K, P là các quân cờ trắng.

Vì số lượng không gian trạng thái phát sinh vô cùng lớn, nên ta không sử dụng cách lưu trữ mỗi nốt con trạng thái là một bàn cờ riêng. Thay vào đó để chuyển trạng thái của bàn cờ từ trạng thái này sang trạng thái khác ta chỉ cần biểu diễn một chuỗi ký tự. Một chuỗi ký tự này bao gồm vị trí bắt đầu, kết thúc và một thông tin cơ bản về loại nước đi của quân cờ (Bao gồm: đi quân, ăn quân, phong tốt, nhập thành). Nước cờ chuyển trạng thái trên phải tuân thủ các quy tắc của cờ vua lập nên (VD: các quy tắc nước đi của quân vua, mã, hậu,….).Từ chuỗi ký tự này, ta tiến hành cài đặt, xử lý để chuyển trạng thái này sang trạng thái khác, chuỗi ký tự này có thể hoạt động hiệu quả trên cả 2 chiều trạng thái *(Mô hình chuỗi ký tự sẽ được giới thiệu rõ ràng ở phần tiếp theo).*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| r | n | b | q | k | b | n | r |
| p | p | p | p | p | p | p | p |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| P | P | P | P | P | P | P | P |
| R | N | B | Q | K | B | N | R |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| r | n | b | q | k | b | n | r |
| p | p | p | p | p | p | p | p |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | P |  |  |  |  |
|  |  |  |  |  |  |  |  |
| P | P | P |  | P | P | P | P |
| R | N | B | Q | K | B | N | R |

Hàm xử lý

Một chuỗi ký tự.

|  |
| --- |
| *Ví dụ: Mô hình chuyển trạng thái 2 chiều từ một chuỗi ký tự (lên tốt trắng 2 bước)* |

Bởi vì quy tắc di chuyển đối với những quân cùng loại của quân cờ đen và quân cờ trắng đều giống nhau (Ví dụ: tốt đen và tốt trắng, xe đen và xe trắng,…), vì thế ta chỉ cần cài đặt xử lý trên quân cờ trắng (là quân in hoa). Ta sẽ cài đặt thêm một mô hình xử lý để chuyển trạng thái bàn cờ từ quân đen sang quân trắng và ngược lại. Sau khi ta đã chuyển được quân đen thành quân trắng thì ta tiếp tục sử dụng mô hình chuyển trạng thái 2 chiều từ một chuỗi ký tự trên quân trắng như phần trên.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| r | n |  | q | k | b | n | r |
| p | p | p |  | p | p | p | p |
|  |  |  |  | b |  |  |  |
|  |  |  | p |  |  |  |  |
|  |  |  | P |  |  |  |  |
|  |  |  |  | P |  |  |  |
| P | P | P |  |  | P | P | P |
| R | N | B | Q | K | B | N | R |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| r | n | b | k | q | b | n | r |
| p | p | p |  | p | p | p | p |
|  |  |  | p |  |  |  |  |
|  |  |  |  | p |  |  |  |
|  |  |  |  | P |  |  |  |
|  |  |  | B |  |  |  |  |
| P | P | P | P |  | P | P | P |
| R | N | B | K | Q |  | N | R |

Hàm xử lý

|  |
| --- |
| *Ví dụ: Mô hình chuyển trạng thái 2 chiều từ quân trắng sang đen và ngược lại.* |

#### Hàm đánh giá

Chúng ta sẽ có một mô hình phân tích thế cờ, mô hình này có thể áp dụng các luật, các phương pháp đánh cờ đã được nghiên cứu và công bố vào từng thế cờ [[13]](http://chessprogramming.wikispaces.com/Evaluation). Nó cho phép ta quy đổi đánh giá của một thế cờ thành một con số nhất định.

Để cho bài toán trở nên thông minh hơn, hàm đánh giá là một trong những yếu tố rất quan trọng tạo ra chọn lựa một nước đi hiệu quả. Để nước đi hiệu quả hơn, qua quá trình tìm hiểu và nghiên cứu, ta chọn được 4 phương pháp có thể áp dụng lại với nhau để tạo ra một mô hình đánh giá hiệu quả, cụ thể như sau:

* Đánh giá dựa trên số quân cờ hiện hữu.
* Đánh giá dựa trên từng vị trí của mỗi loại quân cờ.
* Đánh giá dựa trên số tính cơ động của quân cờ.
* Đánh giá dựa trên độ tấn công của đối thủ.

Điểm số được quy đổi từ thế cờ được sử dụng trong việc tìm kiếm nước đi tối ưu sẽ được giới thiệu ở phần sau.

#### Tìm kiếm

Như đã phân tích độ phức tạp của bài toán, ta sẽ áp dụng thuật toán Alpha-Beta, với khả năng cắt tỉa các trạng thái dư thừa sẽ làm cho kết quả thực thi bài toán sẽ nhanh chóng hơn.

Bài toán sẽ bao gồm 2 chế độ là:

* Máy tính là quân trắng.
* Hoặc máy tính là quân đen.

Ở bất kỳ chế độ nào, thì nước chọn đi ban đầu là Node MAX, nghĩa là:

* Máy tính là quân trắng thì Node MAX nước chọn đi ban đầu sẽ là bàn cờ bắt đầu tuân theo luật của cờ vua.
* Máy tính là quân đen thì Node MAX nước chọn đi ban đầu sẽ phụ thuộc vào nước đi ban đầu của quân đối thủ là con người.

Để tìm kiếm ra nước đi tốt nhất từ Node root MAX trạng thái ban đầu, ta tiến hành theo các bước của thuật toán Alpha – Beta như sau:

* Nếu tại trạng thái đó là đã đụng tới giới hạn của độ sâu, thì ta bắt đầu tính giá trị đánh giá tại vị trí đó
* Nếu tại trạng đó là MIN, ta áp dụng thuật toán cho cá con của nó. Ghi nhớ kết quả Beta.
* Nếu tại trạng đó là MAX, ta áp dụng thuật toán cho cá con của nó. Ghi nhớ kết quả Alpha.

Cuối cùng từ Node root MAX ta tìm được nước đi tốt nhất mà ta có thể đi.

## Cấu trúc dữ liệu và cách biểu diễn các trạng thái của bài toán

### Cách biễu diễn trạng thái của bài toán

Không gian trạng thái của trò chơi cờ vua: là tình thế của bàn cờ được biễu diễn là một biến Board-một ma trận 2 chiều kích thước 8x8, với các phần tử là kiểu String gồm có các giá trị sau: “P”-Tốt trắng, “R”-Xe trắng , “K”-Mã trắng, “B”-Tượng trắng, “Q”-Hậu trắng, “A”-Vua trắng , “p”-Tốt đen , “r”-Xe đen , “k”-Mã đen, , “b”-Tượng đen, “q”-Hậu đen, “a”-Vua đen và “ “-Khoảng trắng xem như là ô trống, không có quân cờ nào ở tại ô đó.

|  |  |  |
| --- | --- | --- |
| **QUÂN CỜ** | **TRẮNG** | **ĐEN** |
| VUA | A | A |
| HẬU | Q | q |
| XE | R | r |
| NGỰA | K | K |
| TƯỢNG | B | b |
| TỐT | P | p |
| Ô trống | “ ” | |

Các phần tử kiểu String đại diện cho mỗi quân cờ

***static*** *String Board[][] = {*

*{****"r"****,****"k"****,****"b"****,****"q"****,****"a"****,****"b"****,****"k"****,****"r"****},  
{****"p"****,****"p"****,****"p"****,****"p"****,****"p"****,****"p"****,****"p"****,****"p"****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****"P"****,****"P"****,****"P"****,****"P"****,****"P"****,****"P"****,****"P"****,****"P"****},  
{****"R"****,****"K"****,****"B"****,****"Q"****,****"A"****,****"B"****,****"K"****,****"R"****},*

*}*

* *Trạng thái ban đầu (root):* vị trí sắp xếp các quân cờ ban đầu trên Board lúc thực hiện giải thuật alphabeta.
* *Trạng thái kết thúc (terminal node):* Khi trên bàn cờ vua, quân Vua của người chơi bị đối thủ của họ chiếu hết. Nhưng vì do không gian trạng thái của cờ vua rất lớn nên việc đi đến trạng thái kết thúc là rất khó thực hiện. Giải pháp được đưa ra là giới hạn độ sâu cho cây tìm kiếm, thay vào đó chúng ta sẽ có các node lá ở độ sâu nhất định là trạng thái được đánh giá
* *Trạng thái được đánh giá (node lá):* được biểu diễn là một Board sử dụng hàm đánh giá tính toán giá trị của Board đó.
* *Toán tử chuyển trạng thái:* là nước đi của một quân cờ. Với mỗi quân cờ khác nhau tương ứng với những nước đi khác nhau. Chúng ta sẽ lập trình các hàm cho từng loại quân cờ để tính toán các nước đi có thể của quân cờ đó với vị trí của nó trên Board (các nước đi này là đúng luật chơi và hợp lệ). Lưu ý: Các hàm tìm nước đi cho quân cờ này chỉ áp dụng trên các quân Trắng, vì vậy để tìm nước đi của các quân Đen cần phải chuyển các quân Đen thành các Quân trắng.

Chúng ta sẽ không biễu diễn các trạng thái dưới dạng dữ liệu có cấu trúc hay là một đối tượng bời vì chúng ta chỉ quan tâm đến giá trị của node-được đánh giá qua bàn cờ của nó. Nếu đó là node lá thì dùng hàm đánh giá để tính giá trị đó, ngược lại thì tùy thuộc node đó thuộc lớp Max hay Min mà lấy giá trị lớn nhất hay nhỏ nhất các node con của nó.

* Chỉ sử dụng duy nhất một biến Board là ma trận 2 chiều kích thước 8x8 có các phần từ là kiểu String (như đề cập ở trên) để biểu diễn bàn cờ vua của các trạng thái.
* Từ 1 trạng thái cha sinh ra các trạng thái con bằng *các toán tử chuyển trạng thái* (nước đi). Vì giải thuật alphabeta mang tính đệ quy, nên để biễu diễn một trạng thái con, ta thực hiện như sau:
  + Trước khi gọi đệ quy đến node con trong giải thuật Alphabeta, chúng ta sử dụng Hàm ApplyMove để thực hiện nước đi lên Board hiện tại đang biễu diễn cho node cha để được Board mới biễu diễn cho node con. Vì giải thuật gồm Trắng và Đen lần lượt ra nước đi, hàm FlipBoard co nhiệm vụ việc thay đổi lượt đi giữa Trắng và Đen.
  + Sau khi gọi đệ quy trạng thái con hoàn tất và quay trở lại node cha, chúng ta sử dụng hàm FlipBoard và UndoMove để trả lại từ Board biểu diễn cho node con thành Board biễu diễn cho node cha từ việc thực hiện nước đi trước khi gọi đệ quy.

flipBoard();

undoMove(String);

FlipBỏad

applyMove(String);

flipBoard();

FlipBỏad

applyMove(String);

flipBoard();

FlipBỏad

flipBoard();

undoMove(String);

FlipBỏad

### Cấu trúc dữ liệu của bài toán

#### Tìm kiếm nước đi tốt nhất

***public static*** *String alphabeta(****int depth****,* ***int alpha****,* ***int beta****, String* ***move****,* ***int player****)*

***static int*** *globalDepth; // độ sâu của cây tìm kiếm*

Đây là hàm cốt lõi của bài toán, với mục đích tìm ra nước đi ngắn nhất. Cấu trúc của hàm alphabeta bao gồm:

**Input:**

* depth (kiểu Int): là giá trị độ sâu.
* alpha (kiểu Int): là giá trị Alpha liên quan đến nút MAX.
* beta (kiểu Int): là giá trị Beta liên quan đến nút MIN.
* move (kiểu String): là một chuỗi gồm 5 ký tự chứa nước đi của node cha khi mở rộng đệ quy sang các node con. Tuy nhiên, trường hợp ngoại lệ là tìm nước đi từ trạng thái ban đầu, nên tham số truyền vào là chuỗi rỗng.
* player (kiểu Int): Máy = MAX = 1, Người = MIN = 0. Để tính, hiện thực, mở rộng các node MAX, MIN tương xứng kiểu người chơi. Và tính giá trị đánh giá cho phù hợp với mỗi node MAX hoặc MIN.
* Sử dụng biến toàn cục - globalDepth. Giới hạn đồ sâu của cây tìm kiếm.

**Output:** Là một chuỗi ký tự gồm 5 ký tự, chứa thông tin của nước tốt nhất mà máy có thể đi. Từ chuỗi ký tự này, qua các hàm xử lý ta có thể sửa đổi bàn cờ và là nước đi cho máy tính.

**Ví dụ về chuỗi Output:**

Một nước di chuyển của quân cờ được biễu diễn là một String có độ dài là 5 ký tự. Nước đi này sẽ được hàm ApplyMove (đã nói phía trên) sử dụng lên biến Board. Hoặc hàm UndoMove để khôi phục lại nước đi trên biến Board.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **String** | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |
| Loại I | Hàng cũ | Cột cũ | Hàng mới | Cột mới | Quân cờ bị ăn |
| Loại II | Cột cũ | Cột mới | Quân cờ bị ăn | Quân cờ được phong | “P” |
| Loại III | Cột cũ của Vua | Cột cũ của Xe | Cột mới của Vua | Cột mới của Xe | “C” |

**VD nước đi loại I:**

a) “3345 ”(chuỗi này có 5 ký tự, ký tự cuối cùng là khoảng trống)

b) “3324k”

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  | k. |  |  | |  |
| 3 |  | |  |  | | P |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | | K |  |  |  | |  |
| 4 |  | |  |  | |  |  | **.** |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

b) Tốt đi từ (3,3) đến ô (2,4) và ăn mã của đối thủ.

a) Mã đi từ (3,3) đến ô trống (4,5)

**VD nước đi loại II:** c) “33 QP” và d) “34bQP”

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | | . |  |  |  | |  |
| 1 |  | |  |  | | P |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  | b. |  |  | |  |
| 1 |  | |  |  | | P |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

d) Tốt đi từ (1,3) đến ô (0,4) và ăn tượng của đối thủ và phong hậu

c) Tốt đi từ (1,3) đến ô trống (0,3) và phong hậu

**VD nước đi loại III**: e) “4765C” và f) “4023C”

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  | A |  |  | | R |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 | R | |  |  | |  | A |  |  | |  |

e) Vua ở ô (7,4) nhập thành gần với Xe ở ô (7,7) sẽ được Vua (7,6) và Xe (7,5)

f) Vua ở ô (7,4) nhập thành xa với Xe ở ô (7,0) sẽ được Vua (7,2) và Xe (7,3)

#### Biểu diễn bàn cờ và các thực hiện nước đi

Biễu diễn bàn cờ cho trạng thái: như đã minh họa ở trên, trong giải thuật alphabeta, thì node cha sẽ gọi đệ quy đến các node con của nó. Nên từ biến Board biểu diễn cho node cha ta sẽ dung hàm applyMove (với tham số là một nước đi có kiểu dữ liệu String gồm 5 ký tự) thay đổi biến Board để biểu diễn cho node con, sau đó gọi hàm flipBoard để đổi lượt cho bên còn lại (chi tiết về hàm flipBoard sẽ được giải thích rõ ràng hơn ở phần sau). Sau khi đệ quy quay trở lại từ node con lên node cha ta cần dùng hàm flipBoard, và undoMove (với tham số là nước đi đã thực hiện trong hàm applyMove trước đó) để biến Board quay trở lại biễu diễn đúng trạng thái của node cha sau đó tiếp tục thực hiện lời gọi đệ quy đến các node khác.

***static*** *String Board[][]; // Biễu diễn trạng thái của bàn cờ*

***public static void*** *flipBoard(); // thay đổi góc nhìn của bàn cờ*

***public static void*** *applyMove(String Move); // thực hiện nước đi lên Board****public static void*** *undoMove(String Move); // khôi phục lại nước đi lên Board*

1. **Biến String Board[][]**

Đây là biến toàn cục, tất cả các di chuyển của quân cờ trên bàn cờ đều thực hiện qua biến này. Nói cách khác biến này lưu trữ trạng thái bàn cờ tại mọi thời điểm.

***static*** *String Board[][] = {*

*{****"r"****,****"k"****,****"b"****,****"q"****,****"a"****,****"b"****,****"k"****,****"r"****},  
{****"p"****,****"p"****,****"p"****,****"p"****,****"p"****,****"p"****,****"p"****,****"p"****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****,****" "****},  
{****"P"****,****"P"****,****"P"****,****"P"****,****"P"****,****"P"****,****"P"****,****"P"****},  
{****"R"****,****"N"****,****"B"****,****"Q"****,****"A"****,****"B"****,****"N"****,****"R"****},*

*}*

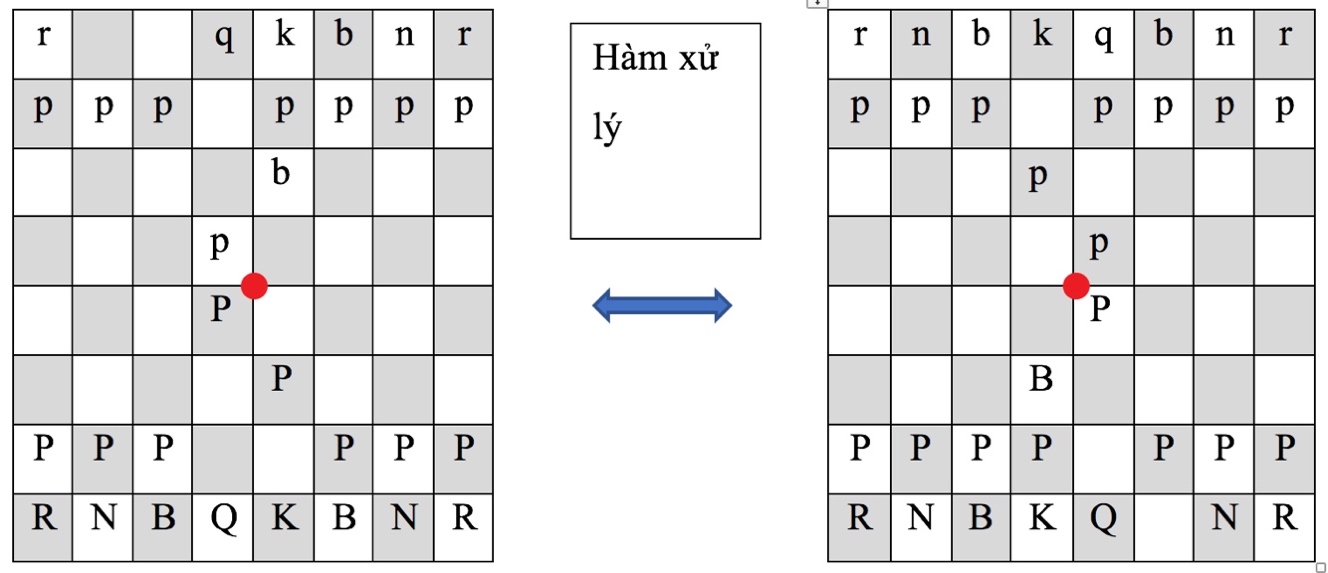
1. **Hàm flipBoard()**

Hàm flipBoard() là hàm xử lý cơ chế thay đổi góc nhìn (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.5) cấu trúc như sau:

**Input:** Xử lý trên biến toàn cục - Board.

**Output:** kiểu void. Kết quả là thay đổi trên biến toàn cục - Board.

**Ví dụ khi gọi hàm này:**



*Trọng tâm*

1. **Hàm applyMove(String Move)**

Hàm applyMove(String Move) là hàm xử lý thực hiện nước đi (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.6.1) cấu trúc như sau:

**Input:** Kiểu String gồm 5 ký tự thể hiện nước đi thực hiện lên Board. (Nước đi có cấu trúc như đã miêu tả ở trên)

**Output**: kiểu void. Kết quả là thay đổi biến toàn cục – Board, thể hiện nước đi được áp dụng lên bàn cờ.

**Ví dụ gọi hàm applyMove(“3324k”):** *// Di chuyển Tốt ăn mã đối thủ.*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  | k |  |  | |  |
| 3 |  | |  |  | | P |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  | P |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

Biến toàn cục Board thay đổi, nghĩa là trạng thái của bàn cờ cũng thay đổi.

1. **Hàm undoMove(String Move)**

Hàm undoMove(String Move) là hàm xử lý phục hồi nước đi (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.6.2) cấu trúc như sau:

**Input:** Kiểu String gồm 5 ký tự thể hiện nước đi thực hiện lên Board. (Nước đi có cấu trúc như đã miêu tả ở trên)

**Output**: kiểu void. Kết quả là thay đổi biến toàn cục – Board, thể hiện nước đi được phục hồi áp dụng lên bàn cờ.

**Ví dụ gọi hàm undoMove(“3324k”):** *// Phục hồi trạng thái từ nước đi tốt ăn mã.*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  | k |  |  | |  |
| 3 |  | |  |  | | P |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  | P |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

Biến toàn cục Board thay đổi, nghĩa là trạng thái của bàn cờ cũng thay đổi.

#### Biểu diễn các nước đi có thể đi

***static int*** *kingPositionU; // vị trí của Vua trắng****static int*** *kingPositionL; // vị trí của Vua đen****static boolean*** *castlingUShort; // kiểm tra nhập thành gần Vua trắng chưa****static boolean*** *castlingULong; // kiểm tra nhập thành xa Vua trắng chưa****static boolean*** *castlingLShort; // kiểm tra nhập thành gần Vua đen chưa****static boolean*** *castlingLLong; // kiểm tra nhập thành xa Vua đen chưa****public static*** *String movePieces(); // trả về kết quả là một chuỗi chứa tất cả các nước đi có thể của một bàn cờ, với mỗi nước đi của quân cờ có độ dài là 5.****public static*** *String moveRook(****int*** *position); // các nước đi có thể của quân xe trên bàn cờ hiện tại****public static*** *String moveQueen(****int*** *position); // các nước đi có thể của quân Hậu trên bàn cờ hiện tại*

***public static*** *String movePawn(****int*** *position); // các nước đi có thể của quân Tốt trên bàn cờ hiện tại****public static*** *String moveKnight(****int*** *position); // các nước đi có thể của quân Mã trên bàn cờ hiện tại*

***public static*** *String moveBishop(****int*** *position); // các nước đi có thể của quân Tượng trên bàn cờ hiện tại****public static*** *String moveKing(****int*** *position); // các nước đi có thể cua Vua trên bàn cờ hiện tại*

***public static boolean*** *safeKing(); // kiểm tra quân Vua có bị chiếu hay ko*

***public static*** *String Set\_GetMove(****int*** *preRow,* ***int*** *preCol,* ***int*** *nextRow,* ***int*** *nextCol); // kiểm tra nước đi có thể thực hiện không và trả về nước đi đó*

1. **Biến KingPositionU**

KingPositionU - kiểu Int (có miền giá trị [0,63]). Vị trí vua của người chơi ra nước đi.

***KingPositionU = Hàng \* 8 + Cột***

Trong đó: Hàng, Cột là vị trí của quân Vua in Hoa (“A”).

1. **Biến KingPositionL**

KingPositionL - kiểu Int (có miền giá trị [0,63]). Vị trí vua của đối thủ.

***KingPositionL = Hàng \* 8 + Cột***

Trong đó: Hàng, Cột là vị trí của quân Vua in Thường(“a”).

1. **Biến castlingUShort**

CastlingUShort - kiểu Boolean. Nhập thành gần của người chơi (có quân cờ là in Hoa – Upper) có thực hiện được hay không (True hoặc False).

1. **Biến castlingULong**

CastlingULong - kiểu Boolean. Nhập thành xa của người chơi (có quân cờ là in Hoa – Upper) có thực hiện được hay không (True hoặc False).

1. **Biến castlingLShort và castlingLLong:** tương tự như 2 biến trên nhưng là nhập thành gần và xa của đối thủ.
2. **Hàm movePieces()**

**Input:** Xử lý trên biến toàn cục - Board.

**Output:** kiểu String. Hàm này trả về giá trị kiểu String là một tập chuỗi các nước đi có thể của tất cả các quân cờ trên biến Board hiện tại. Trong đó, với mỗi nước đi có độ dài 5 ký tự.

Hàm này lấy được tập nước đi từ các hàm, với các hàm là tập nước đi của từng loại quân cờ riêng bao gồm:

* moveRook(int Position).
* moveQueen(int Position).
* movePawn(int Position).
* moveKnight(int Position).
* moveBishop(int Position).
* moveKing(int Position).

Tất cả các hàm tập nước đi của từng loại quân cờ đều có điểm chung như sau:

**Input:** vị trí của quân cờ kiểu Int. Xử lý trên biến toàn cục – Board.

**Output:** là kiểu String thể hiện tất cả nước đi có thể của quân cờ đó. Mỗi nước đi có độ dài 5 ký tự (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.2). Độ dài của String là: số nước đi \* 5.

1. **Hàm safeKing()**

Hàm safeKing() dùng để kiểm tra xem Vua có an toàn (không bị chiếu) hay không.

**Input:** sử dụng biến toàn cục – Board và biến toàn cục - kingPositionU.

**Output:** kiểu Boolean. Kiểm tra xem quân Vua có an toàn với trạng thái hiện tại của biến Board. True: an toàn. False: không an toàn (bị chiếu).

1. **Hàm Set\_GetMove(int preRow, int preCol, int nextRow, int nextCol)**

Hàm Set\_GetMove() thực hiện nước đi từ vị trí cũ đến vị trí mới, với kiểu đi loại I có cấu trúc như sau:

**Input:** là hàng cũ, cột cũ, hàng mới và cột mới của quân cờ đều là kiểu Int. Trong đó sẽ gọi hàm safeKing() để kiểm tra việc thực hiện nước đi có khiến Vua rơi vào tình thế bị chiếu hay không.

**Output:** kiểu String. Nếu nước đi hợp lệ thì trả về nước đi đó – gồm chuỗi 5 ký tự, ngược lại trả về String rỗng.

#### Đánh giá

Việc đánh giá được tính dựa trên cơ sở là lợi thế của người chơi trừ đi lợi thế của đối thủ trên Board hiện tại. Nếu là số dương thì người chơi có lợi thế hơn, nếu số âm thì đối thủ có lợi thế hơn.

***public static int*** *evaluation(****int*** *listLength,* ***int*** *depth); // hàm đánh giá****public static int*** *evalMaterial(); // đánh giá số lượng các quân cờ*

***public static int*** *evalMobility(****int*** *length,* ***int*** *depth); // đánh giá số lượng nước đi****public static int*** *evalAttack(); // đánh giá các đoàn tấn công của đối thủ*

***public static int*** *evalPositional(****int*** *material); // đánh giá vị trí của các quân cờ trên bàn cờ*

***static int*** *BoardPawn[][]; // đánh giá giá trị của quân Tốt ở từng vị trí trên bàn cờ****static int*** *BoardRook[][]; // đánh giá giá trị của quân Xe ở từng vị trí trên bàn cờ****static int*** *BoardKnight[][]; // đánh giá giá trị của quân Ngựa ở từng vị trí trên bàn cờ*  
***static int*** *BoardBishop[][]; // đánh giá giá trị của quân Tượng ở từng vị trí trên bàn cờ****static int*** *BoardQueen[][]; // đánh giá giá trị của quân Hậu ở từng vị trí trên bàn cờ****static int*** *BoardKingMid[][]; // đánh giá giá trị của quân Vua ở từng vị trí trên bàn cờ vào thời điểm giữa ván cờ****static int*** *BoardKingEnd[][]; // đánh giá giá trị của quân Vua ở từng vị trí trên bàn cờ vào thời điểm gần kết thúc ván cờ*

1. **Hàm evaluation(int listLength, int depth)**

Hàm evaluation(int listLength, int depth) là hàm đánh giá chung, sử dụng tổng hợp các phương pháp đánh giá con (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.4) cấu trúc như sau:

**Input:** listLength – kiểu Int: là độ dài của chuỗi nước đi = 5 \* số nước đi có thể đi = độ dài của hàm các nước đi có thể đi movePieces(). Biến depth – kiểu Int: là độ sâu của cây tìm kiếm. Đồng thời sử dụng các điểm số được quy đổi đánh giá bằng các phương pháp đánh giá khác nhau (mô tả phía dưới b, c, d, e).

**Output**: kiểu Int. Kết quả trả về một số điểm đã được quy đổi đánh giá từ một trạng thái của thế cờ.

1. **Hàm evalMaterial()**

Hàm evalMaterial() là là hàm đánh giá dựa trên số quân cờ hiện hữu (được trình bày cụ thể ở các vấn đề thuật giải – 3.3.4.1) cấu trúc như sau:

**Input:** Thực hiện trên biến toàn cục – Board.

**Output**: kiểu Int. Kết quả trả về một số điểm đã được quy đổi bằng phương pháp đánh giá này từ một trạng thái của thế cờ.

1. **Hàm evalMobility(int listLength, int depth)**

Hàm evalMaterial(int listLength, int depth) là hàm đánh giá dựa trên số quân cờ hiện hữu (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.4.3) cấu trúc như sau:

**Input:** listLength – kiểu Int: là độ dài của chuỗi nước đi = 5 \* số nước đi có thể đi = độ dài của hàm các nước đi có thể đi movePieces(). Biến depth – kiểu Int: là độ sâu của cây tìm kiếm.

**Output**: kiểu Int. Kết quả trả về một số điểm đã được quy đổi bằng phương pháp đánh giá này từ một trạng thái của thế cờ.

1. **Hàm evalAttack()**

Hàm evalAttack() là hàm đánh giá dựa trên việc bị tấn công của các quân cờ đối thủ lên các quân cờ của bản thân(được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.4.4) cấu trúc như sau:

**Input:** sử dụng biến toàn cục Board để tính toán.

**Output**: kiểu Int. Kết quả trả về một số điểm đã được quy đổi bằng phương pháp đánh giá này từ một trạng thái của thế cờ.

1. **Hàm evalPositional(int material)**

Hàm evalPositional(int material) là hàm đánh giá dựa trên vị trí của các quân cờ trên bàn cờ (được trình bày cụ thể ở các vấn đề thuật giải – mục 3.3.4.2) cấu trúc như sau:

**Input:**

* material– kiểu Int là giá trị của hàm evalMaterial(): dùng để xác định các thời điểm của ván cờ: đầu, giữa, cuối ván cờ.
* Sử dụng biến toàn cục Board để tính toán.
* Ngoài ra còn có các biến là mảng 2 chiều kiểu Int. Các biến này là dành riêng cho mỗi quân cờ, thể hiện cho vị trí của quân cờ đó tương ứng với giá trị bao nhiêu. Đã được các nhà khoa học nghiên cứu và thống kê mà có được (ở phần đánh giá sẽ nói rõ ràng hơn).
  + Int BoardPawn[][];
  + Int BoardRook [][];
  + Int BoardKnight[][];
  + Int BoardBishop[][];
  + Int BoardQueen[][];
  + Int BoardKingMid[][];
  + Int BoardKingEnd[][];

**Output**: kiểu Int. Kết quả trả về một số điểm đã được quy đổi bằng phương pháp đánh giá này từ một trạng thái của thế cờ.

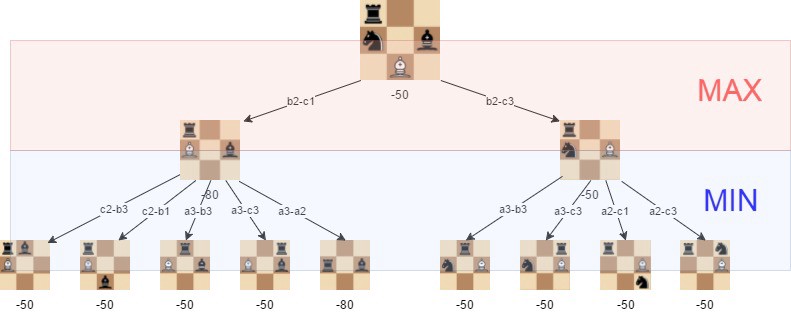
## Các vấn đề của thuật giải

Như đã phân tích ở phần ý tưởng của bài toán, ta sẽ chia các vấn đề của thuật giải theo các mục cụ thể như sau:

### Xây dựng mô hình Alpha-Beta để tìm kiếm

Vì giải thuật Alpha-Beta là giải thuật cải tiến từ giải thuật minimax nên chúng ta sẽ tìm hiểu cơ bản về giải thuật Minimax trước.

**Thuật giải minimax** là giải thuật được sử dụng để xác định nước đi có số điểm tốt nhất có thể trong trò chơi có tổng bằng không sau khi đã nhìn trước một vài nước đi, số điểm đánh giá được tính bằng hàm đánh giá. Giải thuật Minimax đơn giản có thể giải thích như sau: giả sử cây trò chơi sẽ chỉ tìm kiếm có độ sâu là một-chỉ nhìn trước một nước đi, lượt đi của người chơi (Max) đơn giản chỉ là dùng hàm đánh giá để đánh giá những nước đi có thể của mình, nước đi với số điểm cao nhất sẽ được chọn để thực hiện. Nhưng trong độ sâu là hai – nhìn trước 2 nước đi thì sẽ phức tạp hơn bởi đối thủ cũng sẽ thực hiện nước đi, đối thủ (Min) cũng sẽ chọn nước đi mà có điểm số cao nhất cho mình. Vì vậy, điểm số của mỗi nước đi mà Max thực hiện sẽ bằng điểm số tệ nhất của nước đi mà đối thủ đáp trả lại, và sau đó Max sẽ chọn nước đi có điểm số cao nhất trong các nước đi của nó. [*[25]*](https://chessprogramming.wikispaces.com/Minimax)[*[26]*](https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977)



**Thuật giải Alpha-Beta** (cắt tỉa Alpha-Beta hay Alpha-Beta heuristic) là cải tiến từ giải thuật Minimax mà ý tưởng là loại bỏ những node không cần thiết trên một cây trò chơi. Giải thuật bao gồm 2 tham số alpha và beta. Thuật giải này dựa trên trường hợp khi mà chúng ta có thể dừng việc tìm kiếm trên một phần của cây trò chơi nếu chúng ta tìm ra một nước đi dẫn đến một trường hợp tệ hơn nước đi trước đó chúng ta có. Khi sử dụng Thuật giải AlphaBeta, cải tiến từ Minimax, sẽ không làm thay đổi kết quả tìm kiếm của Minimax, ngược lại nó giúp việc tìm kiếm nhanh hơn. Thuật giải AlphaBeta sẽ hiệu quả hơn nếu nó thực hiện việc tìm kiếm trên những nước đi được xem là nước đi hay trước.

**Cách thực hiện:**

Giả sử rằng đến lượt Trắng (Max) đi, chúng ta thực hiện giải thuật ở độ sâu 2 (nghĩa là, chúng ta xét hết tất cả những nước đi của Trắng, và tất cả các nước đi của Đen (Min) đáp trả lại với từng nước đi của Trắng). Đầu tiên chúng ta xét nước đi thứ nhất của Trắng và những nước đi của Đen đáp trả lại (giả sử hàm đánh giá đều trả về cùng giá trị cho tất cả chúng nó). Và nước đi thứ nhất của Trắng sẽ lấy giá trị từ những node con của nó. Sau đó, chúng ta xét nước đi thứ 2 của Trắng và chỉ xét đến nước đi đáp trả đầu tiên của Đen với nước đi này, và chúng ta thấy rằng nếu đi theo thứ tự những nước đi này thì quân Đen sẽ ăn mất quân Tượng Trắng => trong trường hợp này chúng ta sẽ bỏ qua tất cả những nước đi của Đen đáp trả lại lại nước đi thứ 2 của Trắng, bởi vì nước đi thứ nhất của Trắng tốt hơn. Chúng ta thật sự không cần quan tâm nước đi thứ 2 của Trắng dẫn đến kết quả tệ nhất là như thế nào . Sự đánh giá nước đi thứ nhất của Trắng sẽ có giá trị là giới hạn dưới (Alpha), vì vậy bất kỳ những nước đi nào mà đem lại giá trị tệ hơn (nhỏ hơn Alpha) sẽ bị loại bỏ.



Ở trường hợp độ sâu là 3 hoặc sâu hơn, sẽ phức tạp hơn, vì cả 2 người chơi đều có thể đưa ra lựa chọn ảnh hưởng đến trò chơi. Bây giờ chúng ta sẽ giữ cả 2 giá trị là giới hạn dưới và giới hạn trên (gọi là Alpha và Beta). Chúng ta có giới hạn dưới (Alpha) bởi vì nếu có một nước đi khác tệ hơn thì sẽ bỏ qua nó. Và chúng ta có giới hạn trên (Beta) bởi vì ở độ sâu 3 hoặc sâu hơn sẽ có một nước đi tốt hơn cho Max, Min sẽ không cho phép nước đi đó, bởi vì trước đó đã có một nước đi có lợi Min ở độ sâu phía trên trong cây trò chơi mà Min có thể thực hiện để tránh trường hợp này.

**Mã giả:**

***Function*** *alphabeta(depth, alpha, beta, move, player)*

*ListMove = movePieces() //Chuỗi các nước đi có thể đi.*

***if*** *(depth == 0 || ListMove is null)*

***return*** *move + ((-evaluate if player is min) or (evaluate if player is max)) //Player = 1 là MAX. Player = 0 là MIN.*

*ListMove = sortMove(ListMove); //Sắp xếp lại tập các nước đi.*

***for*** *each Move* ***in*** *ListMove: do //Move là một chuỗi gồm 5 ký tự.*

*applyMove(Move); //Thực hiện bước di chuyển*

*flipBoard(); //Đổi góc nhìn. Vì chương trình chỉ . hiện thực trên các quân trắng.*

*value = alphabeta(depth -1, alpha, beta, Move, switch(player)) split (alpha or beta); //Tìm điểm số được đánh . giá từ các node con*

*flipBoard();*

*undoMove(Move); //Thực hiện hồi phục bước di chuyển.*

***if*** *(player is max) //Player = 1 là MAX.*

***if*** *(value > alpha)*

*alpha = value;*

***if*** *depth = globalDepth //Node MAX-Trạng thái ban đầu.*

*return Move; //Đây là nước đi tốt nhất.*

***else*** *// player is min = 0*

***if*** *(value <= beta)*

*beta = value;*

***if*** *(alpha >= beta)*

*break;*

***if*** *(player is max) //Thực hiện ở các node lá.*

*return move and alpha;*

***else***

*return move and beta;*

**Hiện thực code:**

***public static*** *String alphabeta(****int*** *depth,* ***int*** *alpha,* ***int*** *beta, String move,* ***int*** *player){  
 String list = movePieces();* ***if*** *(depth == 0 || list.length() == 0)* ***return*** *move + (rating(list.length(), depth) \* (player \* 2 - 1));  
 list = sortMoves(list);* ***for****(****int*** *i = 0; i < list.length(); i+=5){  
 node++;  
 applyMove(list.substring(i, i + 5));  
 flipBoard();  
 String returnString = alphabeta(depth - 1, alpha, beta, list.substring(i, i+ 5), 1 - player);  
 flipBoard();  
 undoMove(list.substring(i, i + 5));* ***int*** *value = Integer.valueOf(returnString.substring(5, returnString.length()));* ***if*** *(player == 0) {* ***if*** *(value <= beta) {beta = value;}*

*}* ***else****{* ***if*** *(value > alpha) {*

*alpha = value;*

***if*** *(depth == globalDepth)*

*{move = returnString.substring(0,5);}}  
 }* ***if*** *(alpha >= beta) {* ***break****;  
 }  
 }* ***if*** *(player == 0)* ***return*** *(move + beta);* ***else return*** *(move + alpha);  
}*

**Cải thiện:**

Trong giải thuật cắt tỉa Alpha-Beta, số lượng Node được cắt nhiều hay không tùy thuộc vào cách sắp xếp các nước đi trong cây trò chơi. Nếu bạn luôn thực hiện tìm kiếm bất đầu từ những nước đi tốt nhất có thể trước, bạn sẽ cắt bỏ nhiều node nhất. Dĩ nhiên, chúng ta không thể biết được đâu là nước đi tốt nhất trên thực tế (nhưng có thể đánh giá được). Ngược lại nếu bất đầu tìm kiếm từ nước đi tệ nhất, chúng ta sẽ không thể cắt được node nào trên cây trò chơi, nghĩa là nó sẽ thực hiện như giải thuật minimax. Vì lí do đó việc sắp xếp thứ tự nước đi cho việc tìm kiếm là rất quan trọng.

Đã được chứng minh bởi Levin vào nắm 1961, giả sử rằng có b nước đi ở mỗi node và tìm kiếm ở độ sâu n, số lượng node lá tối đa trong cây trò chơi sẽ là b ^ n. Việc tìm kiếm những nước đi tốt nhất trước, sẽ bằng b^ ((n/2)-làm tròn lên) – b ^((n-2)-làm tròn xuống) - 1. [*[27]*](https://chessprogramming.wikispaces.com/Alpha-Beta)

Ta có bảng minh họa sau.

|  |  |  |
| --- | --- | --- |
| **Độ sâu n** | **Số node lá nhiều nhất b^n** | **Số node lá ít nhất**  **b^ ((n/2)-làm tròn lên) – b ^((n/2)-làm tròn xuống) - 1** |
| 0 | 1 | 1 |
| 1 | 40 | 40 |
| 2 | 1,600 | 79 |
| 3 | 64,000 | 1,639 |
| 4 | 2,560,000 | 3,1999 |
| 5 | 102,400,000 | 65,569 |
| 6 | 4,096,000,000 | 127,999 |
| 7 | 163,840,000,000 | 2,623,999 |
| 8 | 6,553,600,000,000 | 5,199,999 |

**Mã giả sắp xếp nước đi từ tốt nhất đến tệ nhất:**

*Function sortMove(ListMove) //ListMove = movePieces()*

1. *i = 0 //Biến đếm, đánh dấu giá trị đánh giá của mỗi bước đi.*
2. ***for*** *each Move* ***in*** *ListMove: do //Tính giá trị mỗi bước đi.*
3. *applyMove(Move)*
4. *Score[i++] = evaluation(-1,0) //Ước lượng đánh giá.*
5. *undoMove(Move)*
6. ***While*** *(ListMove != null) //Sort*
7. *Max = -1000000*
8. ***For*** *each Move in ListMove:* ***do***
9. *Value = evalue each Move*
10. *If (Value > Max)*
11. *Max = Score[Position]//Truy xuất mảng Score.*
12. *Remember this Move*
13. *Move which has max Value is added to newListMove;*
14. *Remove Move which has max Value from ListMove;*
15. *return newListMove;*

**Hiện thực code**

***public static*** *String sortMoves (String list) {****int****[] score=****new int****[list.length()/5];****for*** *(****int*** *i=0;i<list.length();i+=5) {  
 applyMove(list.substring(i,i+5));  
 score[i/5]=evaluation(-1, 0);  
 undoMove(list.substring(i,i+5));  
}  
String newLista=****""****, newListb=list;*

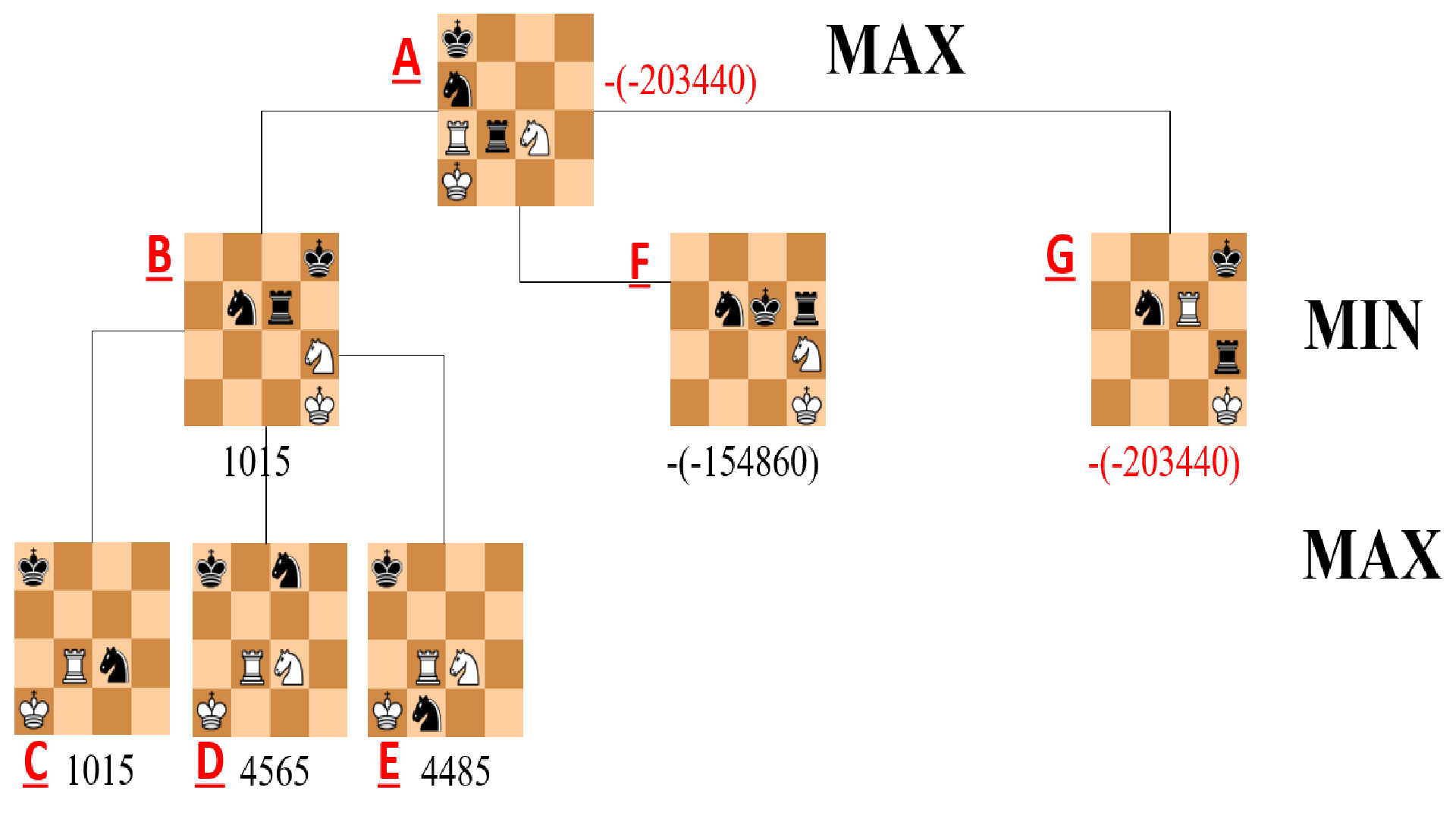
**Ví dụ về hàm alphaBeta:**

***for*** *(****int*** *i=0;i<Math.min(50,list.length()/5);i++) {//only the first few moves are rearanged* ***int*** *max=-1000000,maxLocation=0;* ***for*** *(****int*** *j=0;j<list.length()/5;j++) {* ***if*** *(score[j]>max) {*

*max=score[j];*

*maxLocation=j;*

*}  
 }  
 score[maxLocation]=-1000000;//makes sure it wont be counted again  
 newLista+=list.substring(maxLocation\*5, maxLocation\*5+5);  
 newListb=newListb.replace(list.substring(maxLocation\*5, maxLocation\*5+5),* ***""****);  
}****return*** *newLista+newListb;*

****

Ta có trạng thái ban đầu A (như trên hình), Max-máy tính là quân trắng sẽ thực hiện nước đi. Chúng ta sẽ thực hiện giải thuật AlphaBeta, với depth bằng 2, để tìm nước đi tốt nhất như sau. (lưu ý: chúng ta chỉ tìm 3 nước đi ở mỗi node)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Node | ListMove | Depth | Alpha | Beta | Value |
| A(Max) | “2021r”  “3021r”  “2010k" | 2 |  |  | ?? |
| B(Min) | “6755k”  “6775 ”  “6746 ” | 1 |  |  | ?? |
| C(Max) | Node lá | 0 | Node lá | Node lá | 1015 |
| B(Min) | “6755k”  “6775 ”  “6746 ” | 1 |  | 1015 (1015< | ?? |
| D(Max) | Node lá | 0 | Node lá | Node lá | 4565 |
| B(Min) | “6755k”  “6775 ”  “6746 ” | 1 |  | 1015 (1015<4565) | ?? |
| E(Max) | Node lá | 0 | Node lá | Node lá | 4485 |
| B(Min) | “6755k”  “6775 ”  “6746 ” | 1 |  | 1015 (1015<4485) | **1015** |
| A(Max) | “2021r”  “3021r”  “2010k” | 2 | 1015 (1015>) |  | ?? |
| F(Min) | Node lá | 1(stale-mate) | Node lá | Node lá | 154860 |
| A(Max) | “2021r”  “3021r”  “2010k” | 2 | 154860(154860>  1015) |  | ?? |
| G(Min) | Node lá | 1(checkmate) | Node lá | Node lá | 203440 |
| A(Max) | “2021r”  “3021r”  “2010k” | 2 | 203440 |  | **203440** |

Đang xét ở node: A

* Có độ sâu: 2
* alpha =
* beta =
* Vì node A thuộc lớp MAX nên sẽ tìm node con có giá trị lớn nhất
* ListMove: 2021r3021r2010k

Đang xét ở node: B

* Có độ sâu: 1
* alpha =
* beta =
* Vì node B thuộc lớp MIN nên sẽ tìm node con có giá trị nhỏ nhất
* ListMove: 6755k6775 6746

Đang xét ở node: C

* Có độ sâu: 0
* Đây là node lá và sử dụng hàm đánh giá ta được giá trị: 1015

Đệ quy quay trở lại node: B

* cập nhật lại giá trị beta vì giá trị ở node C(1015) <= beta ()
* giá trị beta cua node B = 1015

Đang xét ở node: D

* Ở độ sâu: 0
* Đây là node lá và sử dụng hàm đánh giá ta được giá trị: 4565

Đệ quy quay trở lại node: B

* giá trị beta vẫn là: 1015 vì 1015 <= giá trị của node D(4565)

Đang xét ở node: E

* Có độ sâu: 0
* Đây là node lá và sử dụng hàm đánh giá ta được giá trị: 4485

Đệ quy quay trở lại node: B

* giá trị beta vẫn là: 1015 vì 1015 <= giá trị của node E(4485)
* Vì đã xét hết node con của B và đã tìm được giá trị nhỏ nhất trong các node con của B nên node B sẽ có giá trị là 1015

Đệ quy quay trở lại node: A

* Cập nhật lại giá trị alpha vì value ở node B(1015) > alpha().
* Nên Alpha của node A = 1015

Đang xét ở node: F

* Có độ sâu: 1
* Đây là node lá bởi vì ở người chơi MIN đã rơi vào tình thế không còn nước đi (checkmate hoặc stalemate). Cụ thể ở trường hợp này là stalemate và sử dụng hàm đánh giá ta được giá trị: 154860

Đệ quy quay trở lại node: A

* Cập nhật lại giá trị alpha vì value ở node F(154860) > alpha(1015)
* Nên alpha của node A= 154860

Đang xét ở node: G

* Có độ sâu: 1
* Đây là node lá bởi vì ở người chơi MIN đã rơi vào tình thế không còn nước đi (checkmate hoặc stalemate). Cụ thể ở trường hợp này là checkmate và sử dụng hàm đánh giá ta được giá trị: 203440

Đệ quy quay trở lại node: A

* Cập nhật lại giá trị alpha vì value ở node G(203440) > alpha(154860)
* Nên alpha của node A= 203440

Cuối cùng giải thuật alphabeta trả về kết quả là move + alpha <=> 2010k203440

### Xây dựng các nước di chuyển có thể đi của quân cờ

Bởi vì thực chất từng loại quân cờ của quân đen và quân trắng đều có nước đi giống nhau, vì thế ta chỉ cần xây dựng nước đi trên một quân trắng. Các nước đi có thể đi từ quân cờ đen thì dùng cơ chế chuyển đổi từ quân đen sang quân trắng và ngược lại ở mục 3.3.1.

Để tiết kiệm không gian lưu trữ của bài toán, muốn tìm các bước di chuyển trạng thái của quân cờ từ vị trí này sang vị trí khác, ta chỉ cần xây dựng một chuỗi ký tự gồm 5 ký tự chứa đầy đủ thông tin di chuyển. Chuỗi ký tự gồm 5 ký sẽ được phân loại thành 3 dạng đáp ứng đầy đủ các luật lệ từ phía trò chơi cờ vua cụ thể như sau:

* Nước cờ di chuyển đến ô trống và ăn quân thông thường. *(Loại I)*
* Nước cờ phong tốt (Quân tốt). *(Loại II)*
* Nước cờ nhập thành (Quân vua và quân xe). *(Loại III)*

Tùy vào từng dạng mà ta có cấu trúc riêng của mỗi dạng như sau:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **String** | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |
| Loại I | Hàng cũ | Cột cũ | Hàng mới | Cột mới | Quân cờ bị ăn |
| Loại II | Cột cũ | Cột mới | Quân cờ bị ăn | Quân cờ được phong | “P” |
| Loại III | Cột cũ của Vua | Cột cũ của Xe | Cột mới của Vua | Cột mới của Xe | “C” |

Tại mỗi vị trí của loại quân đó trên bàn cờ sẽ có nhiều cách di chuyển khác nhau, nghĩa là sẽ có nhiều chuỗi ký tự gồm 5 ký tự. Sao cho với mỗi cách di chuyển vẫn đảm bảo quân vua không bị đối phương ăn.

**Input:** Vị trí của loại quân cờ đó trên bàn cờ (có dạng là một số nguyên, miền giá trị là *0 <= Int <= 63, Int = 8 \* Hàng + Cột* ).

**Output:** Một chuỗi ký tự gồm nhiều cách di chuyển, mỗi cách di chuyển là một chuỗi ký tự gồm 5 ký tự chứa thông tin của nước đi.

Ta sẽ phân tích và tìm hiểu chuỗi ký tự chuyển trạng thái cho việc di chuyển quân cờ ngay sau đây:

#### Quân xe *(Loại I)*

Theo luật cờ vua, quân xe [[14]](https://en.wikipedia.org/wiki/Rook_(chess)) di chuyển theo các đường thẳng dọc theo cột hay hàng tới ô còn trống hoặc tới ô đang bị quân đối phương chiếm giữ mà không bị bất kỳ quân cờ nào cản đường. Ngoại trừ trường hợp, quân xe có thể đi qua quân khác là nhập thành (sẽ được xét ở quân vua).

Ta sẽ thực hiện các bước di chuyển của quân Xe tại một vị trị có thể đi được mà vẫn đảm bảo quân Vua an toàn như sau:

* Quân Xe chỉ đi theo đường thẳng dọc hoặc cột, nghĩa là quân Xe không thay đổi 1 trong 2 giá trị là hàng hoặc cột. Quân Xe di chuyển theo hàng ngang thì giữ lại giá trị giá trị hàng, thay đổi giá trị cột. Ngược lại di chuyển theo hàng dọc thì giữ lại giá trị cột thay đổi giá trị hàng.
* Vì tại đó quân Xe có thể di chuyển được 4 hướng như sau:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | | p• |  |  |  | |  |
| 2 |  | |  |  | | • |  |  |  | |  |
| 3 | • | | • | • | | R | • | • | • | | • |
| 4 |  | |  |  | | • |  |  |  | |  |
| 5 |  | |  |  | | • |  |  |  | |  |
| 6 |  | |  |  | | P |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

* Đi lên (giảm hàng, giữ cột).
* Đi xuống (tăng hàng, giữ cột).
* Sang phải (giữ hàng, tăng cột).
* Sang trái (giữ hàng, giảm cột).
* Như đã phân tích ta sẽ cho 2 biến “*hàng”* và “*cột*” chạy lồng nhau, 2 biến này nhận miền giá trị từ [-1,1]. Tương ứng với các giá trị lần lượt sau:
* -1 : Giảm
* 0 : Không đổi
* 1 : Tăng
* Khi 1 trong 2 giá trị này nhận giá trị 0, thì tương ứng với việc quân Xe di chuyển theo hướng đó. Thế nhưng, giá trị này chỉ đảm bảo quân xe di chuyển được 1 bước. Vì thể ta tạo thêm 1 giá trị là “*khoảng cách”* sẽ tăng lên 1 khi đường đi quân xe không chạm chán với bất kỳ quân nào, *“khoảng cách”* sẽ ngừng gặp khi gặp quân cờ khác. Các bước di chuyển của quân Xe, tới ô trống hoặc tới quân cờ đối phương nếu đảm bảo tính an toàn cho quân Vua thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Xe tại vị trí đó.
* Ta sẽ có bảng sau, trong đó :
* : độ biến thiên vị trí quân Xe theo hàng.
* : độ biến thiên vị trí quân Xe theo cột.
* : giá trị khoảng cách.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Đi lên** | **Đi xuống** | **Qua trái** | **Qua phải** |
|  | -1\**Distance* | 1\**Distance* | 0 | 0 |
|  | 0 | 0 | -1\**Distance* | 1\**Distance* |

**Mã giả:**

***Function*** *move-Rook(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi của quân Xe*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. *Distance = 1 //Số ô mà quân Xe rời đi từ vị trí ban đầu*
5. ***for*** *tempRow = -1 to 1* ***do***
6. ***for*** *tempCol = -1 to 1* ***do***
7. ***if (****tempRow == 0* ***or*** *tempCol == 0)* ***and*** *tempRow != tempCol*
8. *newRow <- Row + Distance \* tempRow*
9. *newCol <- Col + Distance \* tempCol*
10. ***while*** *Board[newRow][newCol] is Blank:* ***do***
11. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing()– Loại I*
12. ***if*** *getMove isn’t null String*
13. *ListMove = ListMove + getMove*
14. *Distance++*
15. *newRow <- Row + Distance \* tempRow*
16. *newCol <- Col + Distance \* tempCol*
    * 1. ***if*** *Board[newRow][newCol] is Lower*
      2. *getMove = Set\_GetMove(row, col, newRow, newCol)*
      3. ***if*** *getMove isn’t null String*
      4. *ListMove = ListMove + getMove*
      5. *Distance = 1*
      6. *Return ListMove*

**Hiện thực code**

***public static*** *String moveRook(****int*** *position){  
 // Return: list has a structure:  
 // normal: [row][col][next row][next col][piece at next row, next col]  
 String list =* ***""****, getMove;* ***int*** *row = position / 8;* ***int*** *col = position % 8;* ***int*** *distance = 1;* ***for*** *(****int*** *i = -1; i <= 1; i++) {* ***for*** *(****int*** *j = -1; j <= 1; j++) {* ***if*** *(i \* j == 0 && i != j) {* ***try*** *{* ***while*** *(****" "****.equals(Board[row + distance \* i][col + distance \* j])) {  
 getMove = Set\_GetMove(row, col, row + distance \* i , col + distance \* j );* ***if****(getMove.length() != 0)  
 {  
 list = list + getMove;  
 }  
 distance++;  
 }*

***if*** *(Character.isLowerCase((Board[row + distance \* i][col + distance \* j]).charAt(0))) {  
 getMove = Set\_GetMove(row, col, row + distance \* i , col + distance \* j );* ***if****(getMove.length() != 0)  
 {  
 list = list + getMove;  
 }  
 }  
 }* ***catch*** *(Exception e) { }  
 distance = 1;  
 }  
 }  
 }* ***return*** *list;  
}*

#### Quân tượng *(Loại I)*

Quân tượng [[15]](https://en.wikipedia.org/wiki/Bishop_(chess)) di chuyển theo các theo đường chéo cùng màu tới ô còn trống hoặc tới ô đang bị quân đối phương chiếm giữ mà không bị bất kỳ quân cờ nào cản đường.

Ta sẽ thực hiện các bước di chuyển của quân Tượng tại một vị trị có thể đi được mà vẫn đảm bảo quân Vua an toàn như sau:

* Quân Tượng chỉ đi chéo trên các ô cùng màu, nghĩa là độ biến thiên vị trí của quân Tượng trên hàng và trên cột đều phải có giá trị tuyệt đối bằng nhau.
* Để cụ thể, tại đó quân Tượng có thể di chuyển được 4 hướng như sau:
* Đi lên sang phải (giảm hàng, tăng cột).
* Đi lên sang trái (giảm hàng, giảm cột).
* Đi xuống sang phải (tăng hàng, tăng cột).
* Đi xuống sang trái(tăng hàng, giảm cột).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  | q | |  |
| 1 |  | | h |  | |  |  | • |  | |  |
| 2 |  | |  | • | |  | • |  |  | |  |
| 3 |  | |  |  | | B |  |  |  | |  |
| 4 |  | |  | • | |  | • |  |  | |  |
| 5 |  | | • |  | |  |  | P |  | |  |
| 6 | • | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

* Như đã phân tích ta sẽ cho 2 biến “*hàng”* và “*cột*” chạy lồng nhau, 2 biến này nhận tập giá trị từ {-1,1}. Tương ứng với các giá trị lần lượt sau:
* -1 : Giảm
* 1 : Tăng
* Khi 2 biến nhận giá trị tương xứng của mình, thì tương ứng với việc quân Tượng di chuyển theo hướng đó. Thế nhưng, giá trị này chỉ đảm bảo quân xe di chuyển được 1 bước. Vì thể ta tạo thêm 1 giá trị là “*khoảng cách”* sẽ tăng lên 1 khi đường đi quân xe không chạm chán với bất kỳ quân nào, *“khoảng cách”* sẽ ngừng gặp khi gặp quân cờ khác. Các bước di chuyển của quân Tượng, tới ô trống hoặc tới quân cờ đối phương nếu đảm bảo tính an toàn cho quân Vua thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Tượng tại vị trí đó.
* Ta sẽ có bảng sau, trong đó :
* : độ biến thiên vị trí quân Tượng theo hàng.
* : độ biến thiên vị trí quân Tượng theo cột.
* : giá trị khoảng cách.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Đi lên sang phải** | **Đi lên sang trái** | **Đi xuống sang phải** | **Đi xuống sang phải** |
|  | -1\**Distance* | -1\**Distance* | 1\**Distance* | 1\**Distance* |
|  | 1\**Distance* | -1\**Distance* | 1\**Distance* | -1\**Distance* |

**Mã giả:**

***Function*** *moveBishop (Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi của quân Tượng*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. *Distance = 1 //Số ô mà quân Xe rời đi từ vị trí ban đầu*
5. ***for*** *each tempRow {-1,1}:* ***do***
6. ***for*** *each tempCol {-1,1}:* ***do***
7. *newRow <- Row + Distance \* tempRow*
8. *newCol <- Col + Distance \* tempCol*
9. ***while*** *Board[newRow][newCol] is Blank:* ***do***
10. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing()– Loại I*
11. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
12. *ListMove = ListMove + getMove*
13. *Distance++*
14. *newRow <- Row + Distance \* tempRow*
15. *newCol <- Col + Distance \* tempCol*
16. ***if*** *Board[newRow][newCol] is Lower*
17. *apply new Bishop Position*
18. ***if*** *safeKing*
19. *ListMove = ListMove + Position*
20. *undo move Bishop Position*
21. *Distance = 1*
22. *Return ListMove*
23. ***if*** *Board[newRow][newCol] is Lower*
24. *getMove = Set\_GetMove(row, col, newRow, newCol*
25. ***if*** *getMove isn’t null String*
26. *ListMove = ListMove + getMove*
27. *undo move Bishop Position*
28. *Distance = 1*
29. *Return ListMove*

**Hiện thực code:**

***public static*** *String moveBishop(****int*** *position){  
 String List =* ***""****, getMove;* ***int*** *row = position / 8,  
 col = position % 8,  
 distance = 1;* ***for*** *(****int*** *tempRow = -1; tempRow <= 1; tempRow += 2)  
 {* ***for*** *(****int*** *tempCol = -1; tempCol <= 1; tempCol += 2)  
 {* ***try*** *{* ***while*** *(****" "****.equals(Board[row + tempRow \* distance][col + tempCol \* distance]))  
 {  
 getMove = Set\_GetMove(row, col, row + tempRow \* distance, col + tempCol \* distance);* ***if*** *(getMove.length() != 0){  
 List = List + getMove;  
 }  
 distance++;  
 }*

***if*** *(Character.isLowerCase(Board[row + tempRow \* distance][col + tempCol \* distance].charAt(0)))  
 {  
 getMove = Set\_GetMove(row, col, row + tempRow \* distance, col + tempCol \* distance);* ***if*** *(getMove.length() != 0){  
 List = List + getMove;  
 }  
 }  
 }****catch*** *(Exception e){}  
 distance = 1;  
 }  
 }* ***return*** *List;  
}*

#### Quân Mã (Loại I)

Quân Mã [[16]](https://en.wikipedia.org/wiki/Knight_(chess)) trong cờ vua đại diện cho hình tượng hiệp sĩ (kỵ binh mặc giáp), mỗi người chơi sẽ bắt đầu ván cờ với 2 quân Mã, và vị trí ban đầu của nó nằm ở giữa Xe và Tượng hay các ô b1, g1 đối với Trắng và b8, g8 đối với Đen.

Nước đi của quân Mã có dạng giống hình chữ L, và có thể di chuyển theo mọi hướng trên bàn cờ. Quân Mã có thể nhảy qua đầu các quân khác, bất kể quân của bên nào, để đi tới điểm đến; và nó là quân duy nhất có khả năng này.

Nước đi của quân Mã có thể di chuyển được 8 hướng như sau:

* Đi hình chéo chữ nhật đứng lên phía trên bên trái (giảm hàng đi 2, giảm cột đi 1).
* Đi hình chéo chữ nhật ngang lên phía trên bên trái (giảm hàng đi 1, giảm cột đi 2).
* Đi hình chéo chữ nhật đứng lên phía trên bên phải (giảm hàng đi 2, tăng cột lên 1).
* Đi hình chéo chữ nhật ngang lên phía trên bên phải (giảm hàng đi 1, tăng cột lên 2).
* Đi hình chéo chữ nhật đứng xuống phía dưới bên trái (tăng hàng lên 2, giảm cột đi 1).
* Đi hình chéo chữ nhật ngang xuống phía dưới bên trái (tăng hàng lên 1, giảm cột đi 2).
* Đi hình chéo chữ nhật đứng xuống phía dưới bên phải (tăng hàng lên 2, tăng cột lên 1).
* Đi hình chéo chữ nhật ngang xuống phía dưới bên phải (tăng hàng lên 1, tăng cột lên 2).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  | **...** |  | **...** |  |  |
| 4 |  |  | **...** |  |  |  | **...** |  |
| 5 |  |  |  |  | K |  |  |  |
| 6 |  |  | **...** |  |  |  | **...** |  |
| 7 |  |  |  | **...** |  | **...** |  |  |
| 8 |  |  |  |  |  |  |  |  |

Ta có ý tưởng tìm nước đi cho quân Mã như sau:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x** | -2 | -2 | -1 | -1 | 1 | 1 | 2 | 2 |
|  | -1 | 1 | -2 | 2 | -2 | 2 | -1 | 1 |

* Nhận xét: Ta thấy rằng vị trí ban đầu của Quân Mã khi đi đến các vị trí mới có sự thay đổi Cột và Hàng lần lượt như bảng sau (x là hàng, y là cột) :

**.**

* Từ nhận xét trên ta có giải pháp là dùng 2 dòng for lồng vào nhau:
  + Dòng for thứ nhất thể hiện x là sự biến thiên của hàng từ [-2,2].
  + Dòng for thứ hai thể hiện y là sự biến thiên của cột từ [-2,2].
  + Vị trí mới sẽ bằng vị trí cũ cộng với sự biến thiên này với điều kiện là
  + Nếu vị trí mới này là vị trí hợp lệ thì return giá trị của hàm là nước đi này. Ngược lại thì return null.

**Mã giả:**

*Function moveKnight(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi của quân Mã*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***For*** *each tempRow [-2, 2]:* ***do***
5. ***For*** *each tempCol [-2, 2]:* ***do***
6. ***If*** *|tempRow \* tempCol| == 2*
7. *newRow <- Row + tempRow;*
8. *newCol <- Col + tempCol;*
9. ***If*** *Board[newRow][newCol] is Lower* ***or*** *Board[newRow][newCol] is blank*
10. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing()*
11. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
12. *ListMove = ListMove + getMove*
13. *Return ListMove*

**Hiện thực Code:**

***public static*** *String moveKnight(****int*** *position){  
 String List =* ***""****, getMove;* ***int*** *row = position / 8,  
 col = position % 8;  
 //Row colume from -2, -1, 1, 2* ***for*** *(****int*** *tempRow = -2; tempRow <= 2; tempRow++)  
 {* ***for*** *(****int*** *tempCol = -2; tempCol <= 2; tempCol++)  
 {* ***if*** *(Math.abs(tempRow \* tempCol) == 2)  
 {* ***try*** *{* ***if****(****" "****.equals(Board[row + tempRow][col + tempCol]) || Character.isLowerCase(Board[row + tempRow][col + tempCol].charAt(0)))  
 {  
 getMove = Set\_GetMove(row, col, row + tempRow , col + tempCol );* ***if****(getMove.length() != 0)  
 {  
 List = List + getMove;  
 }  
 }  
  
 }****catch*** *(Exception e){}  
 }  
 }  
 }* ***return*** *List;  
}*

#### Quân Hậu (Loại I):

Hậu [[17]](https://en.wikipedia.org/wiki/Queen_(chess)) là quân cờ mạnh nhất trong cờ vua. Khi bắt đầu ván cờ, mỗi bên có một Hậu nằm ở hàng đầu cạnh vua. Hậu trắng nằm trên ô trắng và hậu đen nằm trên ô đen. Trong ký hiệu cờ vua, vị trí ban đầu hậu trắng ở d1 và hậu đen ở d8. Funfact: Trên bàn cờ tối đa có thể có 9 quân Hậu (nếu cả 8 quân tốt đều đến được hàng cuối cùng của bàn cờ và được phong Hậu).

Quân Hậu có thể đi ngang, dọc hay chéo tuỳ ý, do đó nó được xem là tổng hợp của [quân xe](https://vi.wikipedia.org/wiki/Xe_(c%E1%BB%9D_vua)) và [quân tượng](https://vi.wikipedia.org/wiki/C%E1%BB%9D_vua#Nguy.C3.AAn_l.C3.BD_ch.C6.A1i). Nếu nó đi vào một ô có quân đối phương thì nó ăn được quân đó.

Quân Hậu sẽ có thể di chuyển được 8 hướng như sau:

* Đi lên (Giữ cột, giảm hàng).
* Đi xuống (Giữ cột, tăng hàng).
* Sang phải (Giữ hàng, tăng cột).
* Sang trái (Giữ hàng, giảm cột).
* Đi lên sang phải (Tăng cột, giảm hàng).
* Đi lên sang trái (Giữ cột, tăng hàng).
* Đi xuống sang phải (Giữ hàng, tăng cột).
* Đi xuống sang trái(Giữ hàng, tăng cột).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | **.** |  |  |  |  |  |  |  |
| 2 |  | **.** |  |  | p. |  |  | **.** |
| 3 |  |  | **.** |  | **.** |  | **.** |  |
| 4 |  |  |  | **.** | **.** | **.** |  |  |
| 5 | **.** | **.** | **.** | **.** | Q | **.** | **.** | **.** |
| 6 |  |  |  | **.** | **.** | **.** |  |  |
| 7 |  |  | **.** |  | **.** |  | **.** |  |
| 8 |  | **.** |  |  | **.** |  |  | **.** |

Như đã phân tích ta sẽ cho 2 biến “*hàng”* và “*cột*” chạy lồng nhau, 2 biến này nhận miền giá trị từ [-1,1]. Tương ứng với các giá trị lần lượt sau:

* -1 : Giảm
* 0 : Không đổi
* 1 : Tăng

Khi cả 2 giá trị này không đồng thời bằng 0, thì tương ứng với việc quân Hậu di chuyển ngang, dọc hay chéo tùy ý. Thế nhưng, giá trị này chỉ đảm bảo quân Hậu di chuyển được 1 bước. Vì thể ta tạo thêm 1 giá trị là “*khoảng cách”* sẽ tăng lên 1 khi đường đi quân Hậu không chạm chán với bất kỳ quân nào, *“khoảng cách”* sẽ ngừng khi gặp quân cờ khác. Các bước di chuyển của quân Hậu, tới ô trống hoặc tới quân cờ đối phương nếu đảm bảo tính an toàn cho quân Vua thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Hậu tại vị trí đó.

Ta sẽ có bảng sau, trong đó :

* : độ biến thiên vị trí quân Hậu theo hàng.
* : độ biến thiên vị trí quân Hậu theo cột.
* : là giá trị khoảng cách.

**(cả không đồng thời bằng 0).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Đi lên** | **Đi xuống** | **Qua trái** | **Qua phải** |
|  | -1\**Distance* | 1\**Distance* | 0 | 0 |
|  | 0 | 0 | -1\**Distance* | 1\**Distance* |
|  | **Đi lên sang trái** | **Đi lên sang phải** | **Đi xuống sang trái** | **Đi xuống sang phải** |
|  | -1\**Distance* | -1\**Distance* | 1\**Distance* | 1\**Distance* |
|  | -1\**Distance* | 1\**Distance* | -1\**Distance* | 1\**Distance* |

**Mã giả:**

*Function moveQueen(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi của quân Hậu*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. *Distance = 1 //Số ô mà quân Xe rời đi từ vị trí ban đầu*
5. ***for*** *each : do*
6. ***for*** *each : do*
7. ***if*** *(!(tempRow == 0 and tempCol ==0))*
8. *newRow <- Row + Distance \* tempRow*
9. *newCol <- Col + Distance \* tempCol*
10. *while Board[newRow][newCol] is blank: do*
11. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing()*
12. *Distance++*
13. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
14. *ListMove = ListMove + getMove*
15. *newRow <- Row + Distance \* tempRow*
16. *newCol <- Col + Distance \* tempCol*
17. *if Board[newRow][newCol] is lower*
18. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing()*
19. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
20. *ListMove = ListMove + getMove*
21. *Distance = 1;*
22. *Return ListMove*

**Hiện thực Code:**

***public static*** *String moveQueen(****int*** *position){  
 // Return: list has a structure:  
 // normal: [row][col][next row][next col][piece at next row, next col]  
 String list =* ***""****, getMove;* ***int*** *row = position / 8, col = position % 8;* ***int*** *distance = 1;* ***for*** *(****int*** *i = -1; i <= 1; i++){* ***for****(****int*** *j = -1; j <= 1; j++){* ***if*** *(i != 0 || j != 0){* ***try****{* ***while****(****" "****.equals(Board[row + distance \* i][col + distance \* j])){  
 getMove = Set\_GetMove(row, col, row + distance \* i , col + distance \* j );* ***if****(getMove.length() != 0)  
 {  
 list = list + getMove;  
 }  
 distance++;  
 }* ***if*** *(Character.isLowerCase(Board[row + distance \* i][col + distance \* j].charAt(0))){  
 getMove = Set\_GetMove(row, col, row + distance \* i , col + distance \* j );* ***if****(getMove.length() != 0)  
 {  
 list = list + getMove;  
 }  
 }  
 }****catch****(Exception e){ }  
 distance = 1;  
 }  
 }  
 }  
}*

#### Quân Tốt (Loại I, Loại II)

Tốt [[18]](https://en.wikipedia.org/wiki/Pawn_(chess)) được xem như là quân yếu nhất trong cờ vua, tuy vậy vai trò của nó rất quan trọng. Mỗi bên người chơi sẽ có 8 quân Tốt, được đặt theo hàng ngang thứ 2 với quân trắng lần lượt có vị trí từ a2, b2,…,h2 và thứ 7 với quân đen lần lượt có vị trí từ a7, b7,…h7.

Quân tốt tùy theo trường hợp mà có các nước đi khác nhau, cụ thể gồm các cách di chuyển sau:

* Quân tốt ở vị trí đầu tiên của nó khi chưa thực hiện nước đi nào thì có thể tiến lên trước 2 bước hay 1 bước tùy úy người chơi (giảm hàng đi 2 hoặc 1; giữ nguyên cột).
* Nếu Tốt trước đó đã di chuyển rồi, ở các nước đi sau đó nó chỉ có thể tiến lên trước 1 ô (giảm hàng đi 1, giữ nguyên cột).
* Quân tốt khi ăn quân của đối thủ thì ăn chéo về 2 phía trái hoặc phải (giảm hàng đi 1; giảm cột đi 1 or tăng cột lên 1).
* Quân tốt có khả năng đặc biệt đó là khi đi đến ô thuộc hàng cuối cùng phía kia bàn cờ nó sẽ được tùy ý phong cấp thành các quân khác như Xe, Mã, Tượng, Hậu (thông thường nó sẽ được phong thành quân mạnh nhất là Hậu).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  | **\*** | **k\*** |  |  |  |
| 1 |  |  |  | **P** |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  | **.** |  |  |  |  |  |
| 5 |  | **b.** | **.** | **k.** |  |  |  |  |
| 6 |  |  | **P** |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

(\*-P sẽ được phong cấp thành 1 trong các quân Xe, Mã, Tượng, Hậu)

Hàm tìm các nước đi có thể của quân Tốt có thể được chia làm các trường hợp sau:

* Ăn quân
* Ăn quân và phong cấp
* Tiến 1 bước
* Tiến 1 bước và phong cấp
* Tiến 2 bước (khi ở vị trí đầu tiên của nó)

**Ăn quân (Loại I):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  | **k.** |  | **q.** |  |  |  |
| 3 |  |  |  | **P** |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

Ta sẽ có bảng sau, trong đó :

* : độ biến thiên vị trí quân Tốt theo hàng.
* : độ biến thiên vị trí quân Tốt theo cột.

|  |  |  |
| --- | --- | --- |
|  | **Ăn quân phía trái** | **Ăn quân phía phải** |
|  | -1 | -1 |
|  | -1 | 1 |

Trong đó, luôn bằng -1, và chỉ có giá trị là -1 và 1. Vì vậy ý tưởng là chúng ta sẽ chỉ dùng 1 dòng for thể hiện cho độ biến thiên của đề tìm nước đi của quân Tốt. Các bước di chuyển của quân Tốt, tới ô trống hoặc tới quân cờ đối phương nếu đảm bảo tính an toàn cho quân Vua thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Tốt tại vị trí đó. (Lưu ý là vị trí của quân Tốt không được thuộc hàng 1, để tránh trường hợp Phong cấp cho tốt khi đi nước tiếp theo, ở trường hợp này chúng ta sẽ có cách giải quyết khác).

**Mã giả**:

*Function movePawn(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***for*** *:* ***do***
5. *newRow <- Row + Distance \* tempRow*
6. *newCol <- Col + Distance \* tempCol*
7. ***if*** *(Board[newRow][newCol] is Lower && Row != 1)*
8. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing()*
9. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
10. *ListMove = ListMove + getMove*

**Ăn quân và phong cấp (Loại II):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  | **q\*** |  | **b\*** |  |  |
| 1 |  |  |  |  | **P** |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

Tương tự như cách ăn quân của Tốt ở trên, nhưng vị trí của Tốt lúc này phải đang ở hàng thứ 1 trên Board. Ý tưởng giải quyết là có 1 mảng tempPiece chứa các quân Xe, Mã, Tượng, Hậu – đây là những quân mà tốt sẽ có thể được phong cấp. Nghĩa là với 1 nước ăn quân của Tốt nó sẽ có thể được phong thành Xe, Mã, Tượng hoặc Hậu.

**Mã giả**:

*Function movePawn(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***for*** *:* ***do***
5. *newRow <- Row + Distance \* tempRow*
6. *newCol <- Col + Distance \* tempCol*
7. ***if*** *(Board[newRow][newCol] is Lower && Row == 1)*
8. ***for*** *each tempPiece = [R,K,B,Q]* ***do***
9. *Apply new Pawn Position //ApplyMove(String)*
10. ***if*** *safeKing()*
11. *ListMove = ListMove + Position //Position là một chuỗi được tạo ra từ các thông tin theo cấu trúc Loại 2*
12. *Undo move Pawn Position //UndoMove(String)*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  | **.** |  |  |  |
| 3 |  |  |  |  | **P** |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

**Tiến 1 bước (Loại I):**

Ta sẽ có bảng sau, trong đó :

* : độ biến thiên vị trí quân Tốt theo hàng.
* : độ biến thiên vị trí quân Tốt theo cột.

|  |  |
| --- | --- |
|  | **Tiến 1 bước** |
|  | -1 |
|  | 0 |

Trong đó, bằng -1, và bằng 0. Các bước di chuyển của quân Tốt, tới ô trống phía trước nếu đảm bảo tính an toàn cho quân Vua thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Tốt tại vị trí đó. (Lưu ý là vị trí của quân Tốt không được thuộc hàng 1, để tránh trường hợp Phong cấp cho tốt khi đi nước tiếp theo, ở trường hợp này chúng ta sẽ có cách giải quyết khác).

**Mã giả**:

*Function movePawn(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***if****(Board[newRow][newCol] is blank && Row != 1)*
5. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing() – Loại I*
6. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
7. *ListMove = ListMove + getMove*

**Tiến 1 bước và phong cấp (Loại II):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  |  | **\*** |  |  |  |
| 1 |  |  |  |  | **P** |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

Tương tự như tiến 1 bước, nhưng vị trí của Tốt lúc này phải đang ở hàng thứ 1 trên Board. Ý tưởng giải quyết là có 1 mảng tempPiece chứa các quân Xe, Mã, Tượng, Hậu – đây là những quân mà tốt sẽ có thể được phong cấp. Nghĩa là với 1 nước ăn quân của Tốt nó sẽ có thể được phong thành Xe, Mã, Tượng hoặc Hậu.

**Mã giả**:

*Function movePawn(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***if****(Board[newRow][newCol] is blank && Row == 1)*
5. ***for*** *each tempPiece = [R,K,B,Q]* ***do***
6. *Apply new Pawn Position //ApplyMove(String)*
7. ***if*** *safeKing()*
8. *ListMove = ListMove + Position //Position là một chuỗi được tạo ra từ các thông tin theo cấu trúc Loại 2*
9. *Undo move Pawn Position //UndoMove(String)*

**Tiến 2 bước (Loại I):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 |  |  |  | **q\*** |  | **b\*** |  |  |
| 1 |  |  |  |  | **P** |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |

Ta sẽ có bảng sau, trong đó :

* : độ biến thiên vị trí quân Tốt theo hàng.
* : độ biến thiên vị trí quân Tốt theo cột.

|  |  |
| --- | --- |
|  | **Tiến 2 bước** |
|  | -2 |
|  | 0 |

Trong đó, bằng -1, và bằng 0. Điều kiện là vị trí của quân Tốt phải ở hàng thứ 6 của Board. Các bước di chuyển của quân Tốt, tới ô trống phía trước nếu đảm bảo tính an toàn cho quân Vua thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Tốt tại vị trí đó.

**Mã giả**:

*Function movePawn(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***if****(Board[newRow][newCol] is blank* ***and*** *Row == 6)*
5. ***for*** *each tempPiece = [R,K,B,Q]* ***do***
6. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing() – Loại I*
7. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
8. *ListMove = ListMove + getMove*
9. *Undo move Pawn Position*

***public static*** *String movePawn(****int*** *position){  
// Return: list has a structure:*

*// normal: [row][col][next row][next col][piece at next row, next col]  
// if pawn promotion : [col] [nex col] [captured piece] [promoted Piece] [P(pawn)]  
 String list =* ***""****, getMove, oldPiece;* ***int*** *row = position / 8, col = position % 8;* ***for****(****int*** *tempCol = -1; tempCol <=1; tempCol+=2){  
 // capture opponent's piece* ***try****{* ***if*** *(Character.isLowerCase(Board[row - 1][col + tempCol].charAt(0)) && row != 1){  
 getMove = Set\_GetMove(row, col, row -1, col+tempCol);* ***if****(getMove.length() != 0){  
 list = list + getMove;  
 }  
 }  
 }****catch****(Exception e){}*

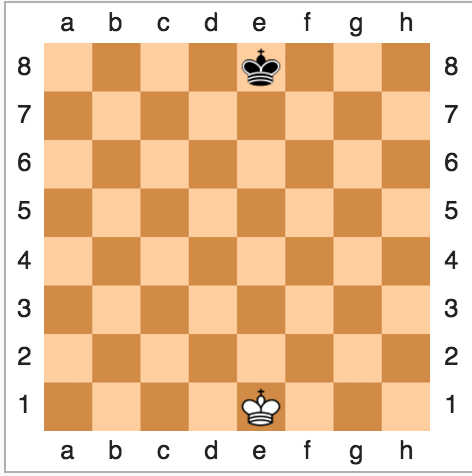
**Hiện thực Code (tất cả các trường hợp):**

*// capture opponent's piece and promote* ***try****{* ***if*** *(Character.isLowerCase(Board[row - 1][col + tempCol].charAt(0)) && row == 1){  
 String temp[] = {****"Q"****,* ***"R"****,* ***"K"****,* ***"B"****};* ***for*** *(****int*** *i = 0; i < 4; i++) {  
 oldPiece = Board[row - 1][col + tempCol];  
 Board[row][col] =* ***" "****;  
 Board[row - 1][col + tempCol] = temp[i];* ***if*** *(safeKing()) {  
 list = list + col + (col + tempCol) + oldPiece + temp[i] +* ***"P"****;  
 }  
 Board[row][col] =* ***"P"****;  
 Board[row - 1][col + tempCol] = oldPiece;  
 }  
 }  
 }****catch****(Exception e){}  
 }  
 //go straight 1 distance* ***try****{* ***if*** *(****" "****.equals(Board[row - 1][col]) && row != 1){  
 getMove = Set\_GetMove(row, col, row -1, col);* ***if*** *(getMove.length() != 0){  
 list = list + getMove;  
 }  
 }  
 }****catch****(Exception e){}  
 //go straight 1 distance and promote* ***try****{* ***if*** *(****" "****.equals(Board[row - 1][col]) && row == 1){  
 String temp[] = {****"Q"****,* ***"R"****,* ***"K"****,* ***"B"****};* ***for*** *(****int*** *i = 0; i < 4; i++) {  
 oldPiece =* ***" "****;  
 Board[row][col] =* ***" "****;  
 Board[row - 1][col] = temp[i];* ***if*** *(safeKing()) {  
 list = list + col + col + oldPiece + temp[i] +****"P"****;  
 }  
 Board[row][col] =* ***"P"****;  
 Board[row - 1][col] =* ***" "****;  
 }  
 }  
 }****catch****(Exception e){}*

*//go straight 2 distance* ***try****{* ***if*** *(****" "****.equals(Board[row - 1][col]) &&* ***" "****.equals(Board[row - 2][col]) && row == 6){  
 getMove = Set\_GetMove(row, col, row -2, col);* ***if*** *(getMove.length() != 0){  
 list = list + getMove;  
 }  
 }  
 }****catch****(Exception e){}* ***return*** *list;  
}*

#### Quân Vua (Loại I và Loại III)

Trong cờ vua, quân Vua [[19]](https://en.wikipedia.org/wiki/King_(chess)) là quân quan trọng nhất, biểu trưng cho quyền lực. Mục tiêu của mỗi người chơi đều phải chiếu hết được quân Vua của đối phương. Đầu cuộc chơi, với số lượng các quân vệ thần lớn, nên vua lại quân cờ yếu nhất (xét về khả năng di chuyển), thế nhưng khi đã tàn cuộc, quân Vua lại đóng vai trò quan trọng trong việc tấn công và hỗ trợ quân Tốt phong cấp.

Khi bắt đầu ván cờ, quân Vua của kỳ thủ cầm quân trắng sẽ đứng ở vị trí thứ 5 tính từ bên trái của người đó, còn quân Vua của kỳ thủ cầm quân đen sẽ ở vị trí thứ 4.

Quân Vua có thể đi đến các ô trống, hoặc có thể ăn quân ở các ô bao quanh mình. Tuy nhiên quân Vua không thể di chuyển đến các ô mà của quân mình đang chiếm giữ và các ô đang bị quân cờ của đối phương kiểm soát.

Cụ thể, quân Vua sẽ đi được các hướng như sau:

* Đi lên (Giữ cột, giảm hàng).
* Đi xuống (Giữ cột, tăng hàng).
* Sang phải (Giữ hàng, tăng cột).
* Sang trái (Giữ hàng, giảm cột).
* Đi lên sang phải (Tăng cột, giảm hàng).
* Đi lên sang trái (Giữ cột, tăng hàng).
* Đi xuống sang phải (Giữ hàng, tăng cột).
* Đi xuống sang trái(Giữ hàng, tăng cột).
* Nhập thành xa.
* Nhập thành gần.

Sau đây, ta sẽ chia cụ thể thành 2 trường hợp để dễ dàng phân tích và thiết kế:

* Những nước đi thông thường
* Nước đi nhập thành

**Những nước đi thông thường: (Loại I)**

Về bản chất, những nước đi thông thường của quân Vua khá giống quân Hậu, chỉ khác ở chỗ quân Vua chỉ đi được một đơn vị khoảng cách bằng 1.

Như đã phân tích ta sẽ cho 2 biến “*hàng”* và “*cột*” chạy lồng nhau, 2 biến này nhận miền giá trị từ [-1,1]. Tương ứng với các giá trị lần lượt sau:

* -1 : Giảm
* 0 : Không đổi
* 1 : Tăng

Khi cả 2 giá trị này không đồng thời bằng 0, thì tương ứng với việc quân Vua di chuyển ngang, dọc hay chéo tùy ý. Các bước di chuyển của quân Vua, tới ô trống hoặc tới quân cờ đối phương nếu đảm bảo tính an toàn cho nó thì sẽ đưa vào chuỗi ký tự các nước đi có thể đi của quân Vua tại vị trí đó.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  | **.** | **.** | **.** |  |  |
| 6 |  |  |  | **.** | **A** | **.** |  |  |
| 7 |  |  |  | **.** | **.** | **.** |  |  |
| 8 |  |  |  |  |  |  |  |  |

Ta sẽ có bảng sau, trong đó :

* : độ biến thiên vị trí quân Vua theo hàng.
* : độ biến thiên vị trí quân Vua theo cột.

**(cả không đồng thời bằng 0)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Đi lên** | **Đi xuống** | **Qua trái** | **Qua phải** |
|  | -1 | 1 | 0 | 0 |
|  | 0 | 0 | -1 | 1 |
|  | Đi lên sang trái | Đi lên sang phải | Đi xuống sang trái | Đi xuống sang phải |
|  | -1 | -1 | 1 | 1 |
|  | -1 | 1 | -1 | 1 |

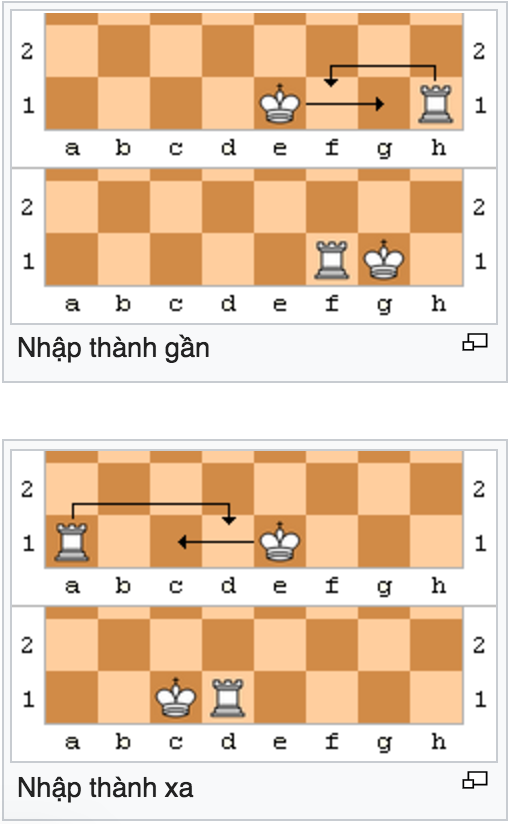
**Mã giả**

***Function*** *moveKing(Position)*

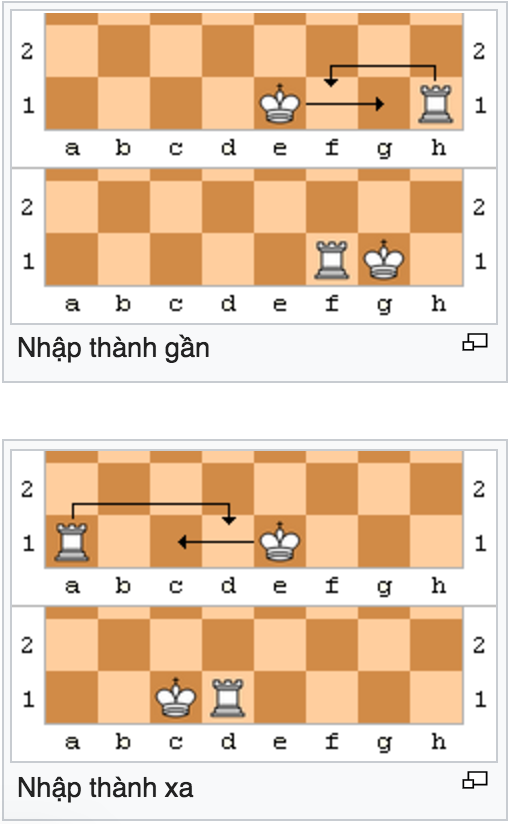
1. *ListMove = “” (null String) //Tập chuỗi nước đi quân Vua*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***for*** *each :* ***do***
5. ***for*** *each :* ***do***
6. ***if*** *(!(tempRow == 0 and tempCol ==0))*
7. *newRow <- Row + Distance \* tempRow*
8. *newCol <- Col + Distance \* tempCol*
9. ***if*** *Board[newRow][newCol] is lower* ***or*** *Board[newRow][newCol] is Blank*
10. *getMove = Set\_GetMove(row, col, newRow, newCol) //Tạo chuỗi di chuyển – Nếu nước đi đó quân Vua an toàn safeKing() – Loại I*
11. ***if*** *getMove isn’t null String //Chuỗi tồn tại*
12. *ListMove = ListMove + getMove*
13. *Return ListMove*

**Nước đi nhập thành:**

Nước đi nhập thành là một loại nước đi rất đặc biệt, giúp tạo thêm khả năng phòng thủ cho quân Vua. Quân Vua cùng với quân Hậu cùng nhau thực hiện nước đi đặc biệt này. Khi thực hiện nhập thành, quân Vua sẽ đi đến ô cùng màu gần nhất, về phía một trong hai quân quân Xe bên mình. Sau đó, quân Xe sẽ được nhảy qua đầu quân Vua. Và cũng theo FIDE – Liên đoàn Cờ vua Quốc tế, khi thực hiện nhập thành, phải thực hiện bằng một tay và phải di chuyển quân Vua trước rồi mới đến quân Xe. Đồng thời, nhập thành cũng đòi hỏi thêm những yêu cầu sau:

* Quân Vua và quân Xe chưa thực hiện bất cứ một bước di chuyển nào.
* Không có quân cờ nào nằm giữa hai quân Vua và Xe.
* Quân Vua không bị chiếu.
* Trong không gian giữa hai quân Vua và Xe, các ô đó không bị đối phương kiểm soát.

Để giải quyết được các yêu cầu kể trên ta sẽ xử lý lần lượt như sau:

* Kiểm tra xem quân Vua và quân Xe chưa thực hiện bất cứ bước di chuyển nào.
* Vùng không gian nằm giữa 2 quân cờ phải là những ô trống.
* Quân Vua phải an toàn (không bị chiếu).
* Giả sử cho quân Vua di chuyển lần lượt giữa các ô trong vùng không gian giữa hai quân Vua và Xe. Mọi bước di chuyển của quân Vua đều phải an toàn (không bị chiếu).

**Mã giả**

***Function*** *moveKing(Position)*

1. *ListMove = “” (null String) //Tập chuỗi nước đi*
2. *Row <- Position / 8*
3. *Col <- Position % 8*
4. ***If*** *CastlingLong* ***and*** *safeKing* ***and (****3 squares between of 2 pieces* ***is*** *Blank) //Tùy vào trường hợp CastlingLong có thể (CastlingULong hoặc CastlingUShort)*
5. *Flag <- true*
6. ***for*** *each i:* ***do*** *//N là vị trí nằm giữa 2 quân – 3 vị trí*
7. *Apply new King Position //ApplyMove(String) với String được tạo ra từ Position loại 3*
8. ***if !****safeKing()*
9. *Flag <- flase*
10. ***break***
11. *Undo move King Position //UndoMove(String)*
12. ***if*** *Flag*
13. *ListMove = ListMove + Position//Position là một chuỗi được tạo ra từ các thông tin theo cấu trúc Loại 3*
14. ***If*** *CastlingShort* ***and*** *safeKing* ***and (****2**squares beetween of 2 pieces* ***is*** *Blank)*
15. *Flag <- true*
16. ***for*** *each i:* ***do*** *//N là vị trí nằm giữa 2 quân – 2 vị trí Apply new King Position*
17. ***if !****safeKing*
18. *Flag <- flase*
19. ***break***
20. *Undo move King Position*
21. ***if*** *Flag*
22. *ListMove = ListMove + Position*
23. *Return ListMove*

**Hiện thực code cho cả 2 trường hợp:**

**public static** String moveKing(**int** position){  
 String List = **""**, getMove;  
 **int** row = position / 8,  
 col = position % 8;  
 **for** (**int** tempRow = -1; tempRow <= 1; tempRow++){  
 **for** (**int** tempCol = -1; tempCol <= 1; tempCol++){  
 **try** {  
 **if** ( !(tempRow == 0 && tempCol == 0) && (**" "**.equals(*Board*[row + tempRow][col + tempCol]) || Character.*isLowerCase*(*Board*[row + tempRow][col + tempCol].charAt(0))))  
 {  
 getMove = *Set\_GetMove*(row, col, row + tempRow, col + tempCol);  
 **if** (getMove.length() != 0){  
 List = List + getMove;  
 }  
 }  
 }**catch** (Exception e){}  
 }  
 }  
 *//Castling upper long* **if** (*castlingULong* && *safeKing*() && **"A"**.equals(*Board*[7][4]) && **"R"**.equals(*Board*[7][0]) && **" "**.equals(*Board*[7][1]) && **" "**.equals(*Board*[7][2]) && **" "**.equals(*Board*[7][3])){  
 **boolean** checkCastling = **true**;  
 *//Colume where King move* **for** (**int** tempCol = 1; tempCol <= 3; tempCol++){  
 getMove = *Set\_GetMove*(7,4,7,tempCol);  
 **if** (getMove.length() == 0){  
 checkCastling = **false**;  
 **break**;  
 }  
 }  
 **if** (checkCastling){  
 *//Colume Previous King, Rook => Next King, Rook* List = List + **"4023C"**;  
 }  
 }

*//Castling upper short* **if** (*castlingUShort* && *safeKing*() && **"A"**.equals(*Board*[7][4]) && **"R"**.equals(*Board*[7][7]) && **" "**.equals(*Board*[7][5]) && **" "**.equals(*Board*[7][6]))  
 {  
 **boolean** checkCastling = **true**;  
 *//Colume where King move* **for** (**int** tempCol = 5; tempCol <= 6; tempCol++)  
 {  
 getMove = *Set\_GetMove*(7,4,7,tempCol);  
 **if** (getMove.length() == 0)  
 {  
 checkCastling = **false**;  
 **break**;  
 }  
 }  
 **if** (checkCastling)  
 {  
 *//Colume Previous King, Rook => Next King, Rook* List = List + **"4765C"**;  
 }  
 }  
 *//Castling lower long when convert from MIN=>Max* **if** (*castlingLLong* && *safeKing*() && **"A"**.equals(*Board*[7][3]) && **"R"**.equals(*Board*[7][7]) && **" "**.equals(*Board*[7][4]) && **" "**.equals(*Board*[7][5]) && **" "**.equals(*Board*[7][6]))  
 {  
 **boolean** checkCastling = **true**;  
 *//Colume where King move* **for** (**int** tempCol = 4; tempCol <= 6; tempCol++)  
 {  
 getMove = *Set\_GetMove*(7,4,7,tempCol);  
 **if** (getMove.length() == 0)  
 {  
 checkCastling = **false**;  
 **break**;  
 }  
 }  
 **if** (checkCastling)  
 {  
 *//Colume Previous King, Rook => Next King, Rook* List = List + **"3754C"**;  
 }  
 }

#### Xây dựng, lấy chuỗi nước đi cho quân cờ loại I.

*//Castling lower long when convert from MIN=>Max* **if** (*castlingLShort* && *safeKing*() && **"A"**.equals(*Board*[7][3]) && **"R"**.equals(*Board*[7][0]) && **" "**.equals(*Board*[7][1]) && **" "**.equals(*Board*[7][2]))  
 {  
 **boolean** checkCastling = **true**;  
 *//Colume where King move* **for** (**int** tempCol = 1; tempCol <= 2; tempCol++)  
 {  
 getMove = *Set\_GetMove*(7,4,7,tempCol);  
 **if** (getMove.length() == 0)  
 {  
 checkCastling = **false**;  
 **break**;  
 }  
 }  
 **if** (checkCastling)  
 {  
 *//Colume Previous King, Rook => Next King, Rook* List = List + **"3012C"**;  
 }  
 }  
 **return** List;  
}

Vì hầu hết trong cách xây dựng code của nước điquân cờ loại I (chiếm chủ yếu) đều có rất nhiều đặc điểm chung với nhau về cấu trúc. Cho nên ta xây dựng, rút ra được chuỗi 5 ký tự nước đi đó.

Về ý tưởng, hàm này sẽ lấy thông tin vị trí cũ và mới của nước đi quân cờ, từ đó thực hiện giả định nước đi. Kiểm tra xem nước đi đó có đảm bảo an toàn cho Vua không. Nếu quân Vua an toàn, trả về chuỗi ký tự nước đi đó. Sau cùng khôi phục lại trạng thái như ban đầu.

**Mã giả**

***Function*** *Set\_GetMove(preRow, preCol, nextRow, nextCol)*

*//1-2-3 tương tự ApplyMove()*

1. *oldPiece = Board[nextRow][nextCol] //Lưu lại quân cờ bị ăn*
2. *Board[nextRow][nextCol] = Board[preRow][preCol] //Di chuyển quân cờ đến vị trí mới*
3. *Board[preRow][preCol] = “ ” //Ô trống*
4. ***if*** *Board[nextRow][nextCol] is WhiteKing //= “A”*
5. *KingPositionU = nextRow \* 8 + nextCol //Nếu quân Vua di chuyển*
6. ***if*** *safeKing() // Nước đi an toàn, gán chuỗi 5 ký tự*
7. *List = preRow + preCol + nextRow + nextCol + oldPieces*

*//1-2-3 tương tự UndoMove()*

1. *Board[preRow][preCol] = Board[nextRow][nextCol]*
2. *Board[nextRow][nextCol] = oldPiece*
3. ***if*** *Board[preRow][preCol] is WhiteKing //= “A”*
4. *KingPositionU = nextRow \* 8 + nextCol*
5. *return List*

**Hiện thực code**

**public static** String Set\_GetMove(**int** preRow, **int** preCol, **int** nextRow, **int** nextCol){  
 String List= **""**, oldPiece;  
 oldPiece = *Board*[nextRow][nextCol];  
 *Board*[nextRow][nextCol] = *Board*[preRow][preCol];  
 *Board*[preRow][preCol] = **" "**;  
 **if**(*Board*[nextRow][nextCol].equals(**"A"**)){  
 *kingPositionU* = nextRow \* 8 + nextCol;  
 }  
 **if** (*safeKing*()){  
 List = List + preRow + preCol + nextRow + nextCol + oldPiece;  
 }  
 *Board*[preRow][preCol] = *Board*[nextRow][nextCol];  
 *Board*[nextRow][nextCol] = oldPiece;  
 **if**(*Board*[preRow][preCol].equals(**"A"**)){  
 *kingPositionU* = preRow \* 8 + preCol;  
 }  
 **return** List;  
}

### Xây dựng cơ chế kiểm tra sự an toàn (bị chiếu) của quân Vua

Để xây dựng được các nước đi có thể có của các quân cờ khác, việc xây dựng cơ chế kiểm tra sự an toàn của quân Vua là một điều cần thiết. Điều đó làm cho mỗi nước đi phải thực sự an toàn, và không đưa người chơi tự đâm đầu vào chỗ chết. Đồng thời cũng giảm thiểu không gian trạng thái của trò chơi đối với những trạng thái không cần thiết.

Ý tưởng: Để kiểm tra xem quân Vua có bị chiếu hay không, ta có thể kiểm tra xem có nước cờ nào trong tập các nước cờ có thể đi của các quân cờ đối thủ đến quân Vua. Dễ hiểu hơn, việc này thực chất là vét cạn các nước đi có thể có của đối thủ. Vì thế, ta sẽ không dùng cách trên sẽ làm cho thời gian thực thi nhiều hơn, ta sẽ chỉ tính đến những nước cờ có thể có của quân đối thủ đến quân Vua. Thực chất việc quân đối thủ đến quân Vua, cũng nghĩa là từ quân Vua đến quân đối thủ, áp dụng các quy luật đi của quân đối thủ. Để thực hiện điều này, ta sẽ thực hiện lại quy trình truy ngược từ vị trí của quân Vua đến vị trí của quân đối thủ sao cho quá trình truy ngược này áp dụng đầy đủ luật cờ của loại quân đối thủ đó. Nếu quá trình truy ngược tìm được đúng loại quân đối thủ, mà loại quân đó ta đang sử dụng luật của nó để áp dụng trên quân Vua, thì tức là quân Vua của ta đang bị chiếu.

Ta sẽ cùng tìm hiểu qua 2 hình minh họa sau:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 |  |  |  |  |  |  |  |  |
| 2 | B |  |  |  |  |  |  |  |
| 3 |  | **.** |  |  |  |  |  | **.** |
| 4 |  |  | **.** |  |  |  | **.** |  |
| 5 |  |  |  | **.** |  | **.** |  |  |
| 6 |  |  |  |  | **A** |  |  |  |
| 7 |  |  |  | **.** |  | **.** |  |  |
| 8 |  |  | **.** |  |  |  | **.** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 |  |  |  |  | **.** |  |  |  |
| 2 |  |  |  |  | **.** |  |  |  |
| 3 |  |  | **r** |  | **.** |  |  |  |
| 4 |  |  |  |  | **.** |  |  |  |
| 5 |  |  |  |  | **.** |  |  |  |
| 6 | **.** | **.** | **.** | **.** | **A** | **.** | **.** | **.** |
| 7 |  |  |  |  | **.** |  |  |  |
| 8 |  |  |  |  | **.** |  |  |  |

* Ở hình bên trái, thực hiện quá trình truy ngược từ vị trí của quân Vua, giả sử quân Vua có cách đi như là quân Tượng. Trong quá trình truy ngược đó, quân Vua có thể tìm đến quân Tượng với luật đi của quân Tượng.
* Ở hình bên phải, giả sử quân Vua có cách đi như là quân Xe. Trong quá trình truy ngược đó, quân Vua không thể tìm đến quân Xe với luật đi của quân Xe.

Để thực hiện được cơ chế trên, ta phải giả sử áp dụng quy tắc đi của các quân của đối thủ lên quân Vua, các cách xây dựng quy tắc đi của các quân cờ đã được giới thiệu ở mục 3.3.2. Cụ thể như sau:

* Quân Tượng và quân Hậu có cùng cách di chuyển chéo.
* Quân Xe và quân Hậu có cùng cách di chuyển thẳng.
* Quân Mã.
* Quân Tốt xét nước đi ăn quân.
* Quân Vua.

***Function*** *safeKing()*

1. ***int*** *RowKing <- kingPositionU / 8*
2. ***int*** *ColKing <- kingPositionU % 8*
3. ***int*** *Distance = 1*

*// let the king move like Bishop and Queen*

1. ***for*** *each :* ***do***
2. ***for*** *each : :* ***do***
3. ***int*** *newRow <- RowKing + Distance \* tempRow*
4. ***int*** *newCol <- ColKing + Distance \* tempCol*
5. ***while*** *Board[newRow][newCol] is blank:* ***do***
6. *Distance++*
7. ***if*** *Board[newRow][newCol] is (queen* ***or*** *bishop) //“q” or “b”*
8. ***return*** *fasle*
9. *Distance = 1*

**Mã giả**

*//Let the King move like Rook and Queen*

1. ***for*** *each :* ***do***
2. ***for*** *each : :* ***do***
3. ***if (****tempRow == 0* ***or*** *tempCol == 0)* ***and*** *tempRow != tempCol*
4. *newRow <- RowKing + Distance \* tempRow*
5. *newCol <- ColKing + Distance \* tempCol*
6. ***while*** *Board[newRow][newCol] is blank:* ***do***
7. *Distance++*
8. ***if*** *Board[newRow][newCol] is (queen* ***or*** *rook)*
9. ***return*** *fasle*
10. *Distance = 1*

*//Let the King move like Knight*

1. ***for*** *each :* ***do***
2. ***for*** *each : :* ***do***
3. ***if*** *|tempRow \* tempCol| == 2*
4. ***if*** *Board[RowKing + ][ColKing + ] is Knight*
5. ***return*** *fasle*

*//Let the King move like Pawn*

1. ***if*** *RowKing >= 2 //Pawn can capture King*
2. ***for*** *each :* ***do***
3. ***if*** *Board[RowKing + ][ColKing + ] is Pawn*
4. ***return*** *fasle*

*//Let the King move like the oponent’s King*

1. ***for*** *each :* ***do***
2. ***for*** *each :* ***do***
3. ***if*** *tempRow != 0 or tempCol != 0*
4. ***if*** *Board[RowKing + ][ColKing + ] is King*
5. ***return*** *fasle*
6. ***return*** *true*

**Hiện thực code**

**public static boolean** safeKing(){  
 **for** (**int** i = 0; i < 64; i++) {  
 **if** (**"A"**.equals(*Board*[i / 8][i % 8])) {  
 *kingPositionU* = i;  
 }  
 **if** (**"a"**.equals(*Board*[i / 8][i % 8])) {  
 kingPositionL = i;  
 }  
 }  
 **int** distance = 1;  
 **int** rowKing = kingPositionU / 8;  
 **int** colKing = kingPositionU % 8;  
 **for** (**int** i = -1; i <=1; i+=2){ *// let the King move like Bishop and Queen* **for** (**int** j = -1; j <= 1; j += 2){  
 **try** {  
 **while** (**" "**.equals(Board[rowKing + distance \* i][ colKing + distance \* j]))  
 {  
 distance++;  
 }  
 **if** (**"b"**.equals(Board[rowKing+ distance \* i][ colKing + distance \* j]) || **"q"**.equals(Board[rowKing + distance \* i][ colKing + distance \* j]))  
 {  
 **return false**;  
 }  
 }  
 **catch** (Exception e) { }  
 distance = 1;  
 }  
 }

*//Rook & Queen go straight* **for** (**int** i = -1; i <= 1; i++) *// let the King move like Rook and Queen* **for**(**int** j = -1; j<=1; j++){  
 **if** (i \* j == 0 && i != j ){  
 **try**{  
 **while**(**" "**.equals(Board[rowKing + distance \* i][ colKing + distance \* j])){  
 distance++;  
 }  
 **if**(**"r"**.equals(Board[rowKing + distance \* i][colKing + distance \* j]) || **"q"**.equals(Board[rowKing + distance \* i][colKing + distance \* j])){  
 **return false**;  
 }  
 }**catch** (Exception e){}  
 distance = 1;  
 }  
 }  
 *//Knight* **for** (**int** i = -2; i<=2; i++){ *// let the King move like Knight* **for** (**int** j =-2; j<=2; j++){  
 **if** (Math.abs(i\*j) == 2){  
 **try**{  
 **if** (**"k"**.equals(Board[rowKing + i][colKing + j])){  
 **return false**;  
 }  
 }**catch** (Exception e){}  
 }  
 }  
 }  
 *//Pawn* **if** (rowKing > 1){  
 **for** (**int** i =-1; i <=1; i+=2){  
 **try**{  
 **if** (**"p"**.equals(Board[rowKing -1][colKing + i])){  
 **return false**;  
 }  
 }**catch** (Exception e){}  
 }  
 }

*//King* **for**(**int** i =-1; i<= 1; i++){  
 **for** (**int** j = -1; j <= 1; j++){  
 **try**{  
 **if**(i != 0 || j!= 0) {  
 **if** (**"a"**.equals(Board[rowKing + i][colKing + j])) {  
 **return false**;  
 }  
 }  
 }**catch**(Exception e){}  
 }  
 }  
 **return true**;  
  
}

### Xây dựng mô hình đánh giá không gian trạng thái

Trải qua hơn 68 năm, ngày Claude Shannon, công bố bài báo đầu tiên về chương trình máy tính trong cờ vua, đó chính là nguồn động lực làm tăng tính thúc đẩy nghiên cứu của các nhà khoa học. Có lẽ sẽ không có những mô hình đánh giá được cho là chính xác tuyệt đối bởi tính phức tạp và chuyên quyền của trò chơi. Thế nhưng những mô hình đánh giá mang tính gần đúng, lại đưa chúng ta đến gần chiến thắng hơn. Mô hình càng phức tạp, được nghiên cứu càng rộng và sâu trong nhiều giải đấu chuyên nghiệp càng làm tăng độ chính xác, giúp bài toán dẫn trở nên thông minh và hiệu quả hơn.

Để đạt hiệu quả của việc xây dựng mô hình đánh giá, có rất nhiều khía cạnh được nghiên cứu, tuy nhiên cơ bản, ta phải xây dựng một mô hình đạt các hiệu quả cụ thể như sau:

* Phát động nước đi.
* Tốt nhất, nên phát động nước đi của các quân cờ từ vị trí ban đầu của chúng.
* Quân Vua nhập thành.
* Các quân Ngựa, Tượng nên chiếm được các vị trí mà có khả năng di chuyển cao.
* Kiểm soát thế trận.
* Chiếm lợi thế ở việc kiểm soát các ô trung tâm của bàn cờ.
* Khả năng cơ động.
* Chọn lựa các vị trí có khả năng di động, di chuyển cao, và không bị hạn chế di chuyển bởi quân mình.
* Cặp Tượng ban đầu là khả năng mở được hai ô màu trên bàn cờ.
* Vua phải luôn an toàn ở đầu và giữa ván cờ.
* Sức mạnh của quân Tốt.
* Giảm thiểu khả năng doubled và tripled (là hai hoặc ba quân tốt cùng trên một cột) của quân Tốt bởi vì chúng không thể bảo vệ cho nhau.
* Giảm thiểu tình trạng quân Tốt bị cô lập, cần được bảo vệ chung với nhau.
* Nên tạo thuận lợi cho quân Tốt qua đường, bởi nó không sợ quân Tốt đối thủ và có cơ hội lớn được phong Hậu.
* Có đủ 8 quân Tốt không phải là ý tưởng tốt, đôi khi ta cần hi sinh ít nhất một đường để mở rộng đường đi của quân Xe.

Đã có rất nhiều bài báo công bố nghiên cứu về vấn đề này của các nhà khoa học, kỳ thủ, mọi người đều mang cho mình những khía cạnh hợp lý. Tuy nhiên nhìn điểm chung của chúng tồn tại nhiều phương pháp cơ bản hiện hữu trong mô hình mà ta nên áp dụng cho việc đánh giá, cụ thể như sau:

* Đánh giá dựa trên số quân cờ hiện hữu. (Material Balance)
* Đánh giá dựa trên từng vị trí của mỗi loại quân cờ. (Piece-square value)
* Đánh giá dựa trên số tính cơ động của quân cờ. (Mobility Score)
* Đánh giá dựa trên độ tấn công của đối thủ. (Attack Score)

Tổng hợp của 4 phương pháp trên ta thành được một mô hình đánh giá Evaluation [[13]](http://chessprogramming.wikispaces.com/Evaluation):

Trong đó:

* Who2Move là lượt di chuyển của quân cờ. Who2Move = +1 là đến lượt quân Trắng di chuyển, Who2Move = -1 là lượt quân Đen.

Với mỗi phương pháp, ta sẽ tìm hiểu kỹ hơn ở phần dưới với đầy đủ công thức về mặt lí thuyết. Tuy nhiên, như đã có giới thiệu ở phần trước về cơ chế thay đổi góc nhìn chỉ cần thực hiện trên quân Trắng, ta sẽ sử dụng cơ chế này để hiện thực mã giả đồng thời hiện thực code một cách đơn giản hơn.

Ta vẽ lại mô hình cây như sau:

applyMove(String);

flipBoard();

FlipBỏad

flipBoard();

undoMove(String);

FlipBỏad

applyMove(String);

flipBoard();

FlipBỏad

flipBoard();

undoMove(String);

FlipBỏad

flipBoard();

undoMove(String);

FlipBỏad

applyMove(String);

flipBoard();

FlipBỏad

applyMove(String);

flipBoard();

FlipBỏad

flipBoard();

undoMove(String);

FlipBỏad

Vì bản thân cây tìm kiếm chọn nốt root MAX là trạng thái ban đầu, dù ở chế độ là máy tính là quân Trắng, hay máy tính là quân Đen, chỉ khác nhau ở nút root.

Giả sử theo như lý thuyết, ta chưa áp dụng flipBoard vào. Bàn cờ đi lần lượt các quân trắng và đen thì hàm đánh giá được tính sau khi chạm đến giới hạn độ sâu như sau:

* Với mỗi nút MAX. Thì sẽ đến lượt quân trắng đi. Tức là
* Với mỗi nút MIN. Thì sẽ đến lượt quân đen đi. Tức là

Ta sẽ áp dụng cơ chế thay đổi góc nhìn (flipBoard) vào, để tính xem Evaluation khi các nút MIN và MAX đến giới hạn độ sâu sẽ thay đổi như thế nào:

* Với mỗi nút MAX. Thì sẽ đến lượt quân trắng đi. Tức là sau khi flipBoard ở nốt MAX đó, bàn cờ đúng với hiện trạng ban đầu, đúng với góc nhìn và màu sắc của quân cờ. Vì thế lúc đó:
* Với mỗi nút MIN. Thì sẽ đến lượt quân đen đi. Tức là sau khi flipBoard ở nốt MIN đó, bàn cờ bị đổi ngược góc nhìn để chuyển đổi nước đi cho quân đen(lúc này quân đen đã thành quân trắng). Vì đây là nước đi của quân đen nên Who2Move = - 1. Đồng thời, vì đã đổi thay đổi góc nhìn, đổi quân đen thành quân trắng và ngược lại, vì thế giá trị tổng Eval sẽ là nghịch đảo của đúng trật tự cờ màu sắc ban đầu. Vì thế lúc đó:

**Mã giả**

***Function*** *evaluation(listLength, depth)*

1. *Eval = evalMaterial()*
2. *Eval += evalPositional(Eval) //Eval ở đây là evalMaterial().*
3. *Eval += evalAttack()*
4. *Eval += evalMobility(listLength, depth)*
5. *Change FlipBoard() //Tính Eval của đối thủ*
6. *Material = evalMaterial()*
7. *Eval -= Material*
8. *Eval -= evalPositional(Material)*
9. *Eval -= evalAttack()*
10. *Eval -= evalMobility(listLength, depth)*
11. *Change FlipBoard() //Lật bàn cờ lại như ban đầu.*
12. *Return Eval*

**Hiện thực code**

**public static int** evaluation(**int** listLength, **int** depth){  
 **int** score = 0;  
 **int** material = *evalMaterial*();  
 score += material;  
 score += *evalPositional*(material);  
 score += *evalAttack*();  
 score += *evalMobility*(listLength, depth);  
 *flipBoard*();  
 material = *evalMaterial*();  
 score -= material;  
 score -= *evalPositional*(material);  
 score -= *evalAttack*();  
 score -= *evalMobility*(listLength, depth);  
 *flipBoard*();  
 **return** score + depth \* 50;  
}

#### Đánh giá dựa trên số quân cờ hiện hữu (Material Balance).

Tên gọi tiếng Anh của phương pháp này là Material Balance. Được giới thiệu lần đầu tiên bởi Shannon [[12]](http://archive.computerhistory.org/projects/chess/related_materials/text/2-0%20and%202-1.Programming_a_computer_for_playing_chess.shannon/2-0%20and%202-1.Programming_a_computer_for_playing_chess.shannon.062303002.pdf) vào năm 1949, khi ấy qua nhiều lần đánh thực nghiệm của bản thân và nghiên cứu ông đã cho ra cấu trúc. Cấu trúc đó như sau:

f(p) = 200(K-K')

+ 9(Q-Q')

+ 5(R-R')

+ 3(B-B' + N-N')

+ 1(P-P')

- 0.5(D-D' + S-S' + I-I')

+ 0.1(M-M') + ...

KQRBNP = number of kings, queens, rooks, bishops, knights and pawns

D,S,I = doubled, blocked and isolated pawns

M = Mobility (the number of legal moves)

Từ cấu trúc trên đã có rất nhiều người tham gia nghiên cứu và kiểm nghiệm. Trong những năm gần đây, cuốn sách “Chess Life” của Larry Kaufman, khiến nhiều người quan tâm hơn cả. Kỳ thủ Kaufman đã có một cuộc nghiên cứu trên 300.000 ván cờ [[20]](http://en.chessbase.com/post/the-joys-of-che-and-the-value-of-the-pieces) của những kiện tướng cờ vua – theo xếp loại của FIDE, với hệ số Elo thấp nhất là 2300. Kết luận của ông, được tổng hợp dựa trên những phân tích đánh giá về sức mạnh lợi thế của quân cờ, sự trao đổi quân cờ nhằm đem lại những nước đi có lợi. Các giá trị điểm của các quân cờ, được đánh giá dựa trên giá trị đơn vị của quân Tốt. Với số điểm lợi thế của tổng điểm các quân cờ càng cao, càng mang lại cho người chơi tỉ lệ thắng cao hơn. Đã có nhiều giá trị của quân cờ được công bố, cụ thể như bảng dưới. Trong bài toán, ta quyết đi chọn đánh giá của Larry Kaufman được công bố năm 2012. Những phân tích của Kaufman, nó vừa mang tính tin cậy vừa dựa vào kết quả của việc phân tích dữ liệu. Phương pháp thống kê của ông cho phép ông giải phóng số lượng lớn dữ liệu từ trò chơi thành những giá trị trung bình điểm số của quân cờ. Vì sử dụng nhiều ván cờ chất lượng, kết quả phân tách được khá chính xác và độ tin cậy cao [[21]](http://chessprogramming.wikispaces.com/Point+Value).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tên người công bố** | **Năm** | **Tốt** | **Mã** | **Tượng** | **Xe** | **Hậu** |
| H. S. M. Coxeter | 1940 | 100 | 300 | 350 | 550 | 1000 |
| Claude Shannon | 1999 | 100 | 300 | 300 | 500 | 900 |
| Alan Turing | 1953 | 100 | 300 | 350 | 500 | 1000 |
| Mac Hack | 1967 | 100 | 325 | 350 | 500 | 975 |
| Chess 4.5 | 1977 | 100 | 325 | 350 | 500 | 900 |
| Tomasz Michniewski | 1995 | 100 | 320 | 330 | 500 | 900 |
| Hans Berliner | 1999 | 100 | 325 | 333 | 510 | 880 |
| Lary Kaufman | 1999 | 100 | 325 | 325 | 500 | 975 |
| Fruit and others | 2005 | 100 | 400 | 400 | 600 | 1200 |
| **Larry Kaufman** | **2012** | **100** | **350** | **350** | **525** | **1000** |

Ta chọn bảng giá trị của Larry Kaufman để đánh giá.

Giá trị của quân Vua sẽ không được cập nhật vào bảng này, bởi vì giá trị của Vua là vô hạn, nhiệm vụ của mỗi người chơi đều phải luôn bảo vệ, gìn giữ quân Vua để đi đến chiến thắng cuối cùng.

Giá trị của Material Balance có thể tính đơn giản bởi công thức sau:

Trong đó:

* i : là số thư tự biểu diễn loại cho quân cờ theo thứ tự của bảng trên. i=0 là Tốt, i=1 là Mã,….
* là gía trị số lượng quân cờ loại i màu Trắng.
* là gía trị số lượng quân cờ loại i màu Đen.
* là gía trị điểm của quân cờ loại i.

Tại mỗi giá trị trạng thái, Material Balance nhận được 1 trong 3 giá trị sau:

* MB > 0 tức là lợi thế đang nghiêng về quân Trắng.
* MB = 0 lợi thế đang cân bằng.
* MB < 0 tức là lợi thế đang nghiêng về quân Đen.

Ngoài ra, ta sử dụng thêm việc tăng thêm điểm cho MB, nếu trạng thái đó tồn tại nhiều hơn một quân Tượng của cùng một màu. Bởi vì với việc nhiều hơn hai quân Tượng, nó có thể kiểm soát được tốt hơn dãy màu mà nó đang chiếm giữ. Ta sẽ thưởng thêm 50 điểm cho số lượng quân Tượng lớn hơn 1.

**Mã giả**

***Function*** *evalMaterial()*

1. ***for*** *each Position*
2. ***switch*** *Board[Position/8][Position%8]: do*
3. ***case*** *WhitePawn: do Score += ValuePawn //WhitePawn = “P”*
4. ***case*** *WhiteRook: do Score += ValueRook //= “R”*
5. ***case*** *WhiteKnight: do Score += ValueKnight // = “K”*
6. ***case*** *WhiteQueen: do Score += ValueQueen // = “Q”*
7. ***case*** *WhiteBishop: do Score += ValueBishop // = “B”*
8. ***if*** *Count(White Bishop) >= 2: do Score += ValueBonus*

*//Các giá trị Value của các quân cờ được lấy từ bảng trên của Larry Kaufman.*

1. *Return Score*

**Hiện thực code:**

**public static int** evalMaterial() {  
 **int** score = 0;  
 **int** countBishop = 0;  
 **for** (**int** position = 0 ; position < 64; position++) {  
 **if** (!**" "**.equals(*Board*[position /8 ][ position % 8])) {  
 **switch** (*Board*[position / 8][position % 8]) {  
 **case "P"**:  
 score += 100;  
 **break**;  
 **case "R"**:  
 score += 525;  
 **break**;  
 **case "K"**:  
 score += 350;  
 **break**;  
 **case "B"**:  
 score += 350;  
 **break**;  
 **case "Q"**:  
 score += 1000;  
 **break**;  
 }  
 }  
 }  
 **if** (countBishop > 1) score += 50 \* countBishop;  
 **return** score \* 7;   
}

#### Đánh giá dựa trên từng vị trí của bàn cờ (Piece-square value).

Có rất nhiều khía cạnh nghiên cứu về vị trí của mỗi quân cờ đứng đâu trên bàn cờ. Mặc dù nó không thể chính xác tuyệt đối vào việc quyết định vị trí của mỗi quân, tuy nhiên cũng cải thiện được phần nào độ thông minh của bài toán. Ta sẽ bắt đầu nghiên cứu về vị trí đứng của mỗi quân, góp phần tạo nên những điểm số tại vị trí đó. Bài viết này được tham khảo tại các diễn đàn cờ vua [[22]](http://chessprogramming.wikispaces.com/Simplified+evaluation+function), bài báo [[23]](http://www.ke.tu-darmstadt.de/publications/reports/tud-ke-2008-07.pdf), phương pháp này đã được mọi người bắt đầu trao đổi rất nhiều vào những năm 1996 khi DeepBlue – trí tuệ nhân tạo Cờ Vua bắt đầu nở rộ. Tên của chủ đề đánh giá này là Mobility Score.

Ta sẽ tìm hiểu một chút sơ lược về việc đánh giá vị trí của mỗi quân cờ cụ thể như sau:

**Quân Tốt:**

// pawn

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

Đối với quân Tốt, ta khuyến khích quân này đi chiếm được những ô trung tâm ở những nước đi đầu tiên. Những việc chiếm được những vị trí ô trung tâm của bàn cờ sẽ làm cho việc hạn chế nước đi của đối thủ. Khi quân Tốt đi càng gần đến những ô cuối cùng của đối thủ, thì chúng càng mạnh, bởi khả năng phong Tốt và quân Tốt qua đường, kết liễu trận đấu.

**Quân Mã:**

Đối với quân Mã, ta thường nên cho những lợi thế của quân này đến những ô trung tâm, bởi vì sự phát động nhiều nước đi hơn. Với các vị trí gần góc mép là một ý tưởng tồi, bởi không thực sự tận dụng được sức mạnh quân Mã. Người ta nói, trong cờ Vua, vị trí đứng sai, thì hậu quả sẽ rất khó lường.

// knight

-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,

**Quân Tượng**

// bishop

-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,

Quân Tượng cũng nên tránh các vị trí ở cạnh bàn, khuyến khích ở các vị trí trung tâm, tuy nhiên cần tránh các mà có thể lên mã của đối thủ.

**Quân Xe Quân Hậu**

//rook

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

//queen

-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

**Quân Vua đầu và giữa trận**

king middle game

-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

Giữa và đầu trận đấu, quân Vua có xu thế đứng sau và nhận được sự che chở của các quân Tốt. Đồng thời, thời điểm này vua rất yếu, nên có xu hướng nhập thành để nhận được sự an toàn cho mình.

**Quân Vua cuối trận**

// king end game

-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

Càng về cuối trận, quân Vua càng mạnh với khả năng có thể bảo vệ được Tốt để phong cấp. Đồng thời với việc di chuyển ở ô trung tâm sẽ làm cho khả năng di chuyển của nó trở nên linh hoạt hơn.

Công thức tính giá trị này là

Trong đó:

* i : là quân cờ trắng.
* j : là quân cờ đen.
* V\_White: Value of Piece White là giá trị theo bảng của quân cờ trắng.
* V\_Black: Value of Piece Black là giá trị theo bảng của quân cờ đen.

Ta còn có thể áp dụng khả năng di chuyển của quân Vua ở các thời điểm là một thời thế, tức là về cuối ván cờ. Khi vua có khả năng di chuyển cao hơn thì điểm nhận được cao hơn.

**Mã giả**

***Function*** *evalPosition(Material) //Material chính là evalMaterial()*

1. ***for*** *each Position*
2. ***switch*** *Board[Position/8][Position%8]: do //Các giá trị của các trường hợp switch case tương tự như ở mã giả evalMaterial()*
3. ***case*** *WhitePawn: do Score += ValuePawn[Position/8][Position%8]*
4. ***case*** *WhiteRook: do Score += ValueRook[Position/8][Position%8]*
5. ***case*** *WhiteKnight: do Score += ValueKnight[Position/8][Position%8]*
6. ***case*** *WhiteQueen: do Score += ValueQueen[Position/8][Position%8]*
7. ***case*** *WhiteBishop: do Score += ValueBishop[Position/8][Position%8]*
8. ***case*** *WhiteKing: do*
9. ***if*** *Material > 1800 //Ở đầu đến giữa ván cờ*
10. *Score += ValueKingMid[Position/8][Position%8]*
11. *Score += moveKing(Position) \* 10 //Giá trị = 10; Quân Vua có nhiều cách di chuyển thì là lợi thế*
12. ***else*** *//Cuối ván cờ*
13. *Score += ValueKingEnd[Position/8][Position%8]*
14. *Score += moveKing(Position) \* 30 //Giá trị = 30; Vào cuối ván cờ, điểm số đánh giá nước đi Vua phải càng cao.*

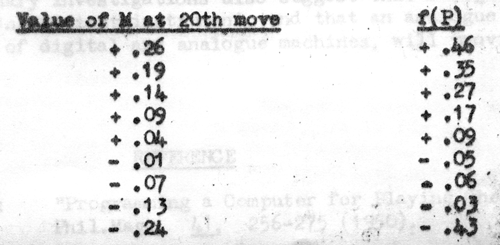
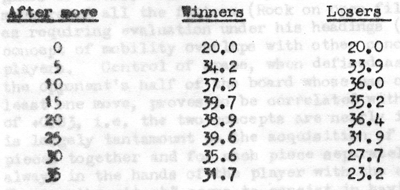
*//Các giá trị của các quân cờ được tính dựa trên giá trị của nó trên từng bàn cờ của quân cờ đó, được giói thiệu các bảng ở trên.*

1. *Return Score*

**Hiện thực code**

**public static int** evalPositional(**int** material) {  
 **int** score = 0, row, col;  
 **for** (**int** position=0; position < 64; position++) {  
 row = position / 8;  
 col = position % 8;  
 **if** (!**" "**.equals(*Board*[row][col])) {  
 **switch** (*Board*[row][col]) {  
 **case "P"**:  
 score += *BoardPawn*[row][col];  
 **break**;  
 **case "R"**:  
 score += *BoardRook*[row][col];  
 **break**;  
 **case "K"**:  
 score += *BoardKnight*[row][col];  
 **break**;  
 **case "B"**:  
 score += *BoardBishop*[row][col];  
 **break**;  
 **case "Q"**:  
 score += *BoardQueen*[row][col];  
 **break**;  
 **case "A"**:  
 **if** (material >= 1800) {  
 score += *BoardKingMid*[row][col];  
 score += *moveKing*(*kingPositionU*).length() \* 10;  
 }  
 **else** {  
 score += *BoardKingEnd*[row][col];  
 score += *moveKing*(*kingPositionU*).length() \* 30;  
 }  
 **break**;  
 }  
 }  
 }  
 **return** score;  
}

#### Đánh giá dựa trên tính cơ động quân cờ (Mobility Score)

Một phương pháp đánh giá dựa trên số các nước đi của quân cờ. Mobility Score là một thuật ngữ để xác định trong chức năng đánh giá quân cờ. Phương pháp dựa trên ý tưởng, bạn có nhiều nước di chuyển hơn thì bạn có cơ hội chiến thắng cao hơn. Một thống kê của Eliot Slater [[24]](http://www.eliotslater.org/index.php/chess/147-statistics-for-the-chess-computer-and-the-factor-of-mobility) (một nhà khoa học tâm lý học nổi tiếng của nước Anh – với sở thích đam mê cờ), với hơn 350 trận đấu chuyên nghiệp , đã chỉ ra rằng sau nước đi thứ 20, nếu Material Balance là cân bằng giữa 2 kỳ thủ, thì tại thời điểm đó nếu kỳ thủ nào sở hữu nhiều nước đi hơn thì cơ hội chiến thắng cao hơn.

Sơ lược, công thức tính giá trị khá đơn giản là

Trong đó:

* : là số cách di chuyển của quân cờ trắng.
* : là số cách di chuyển của quân cờ trắng.
* **V :** là giá trị đánh giá của quân cờ.

Ngoài ra, Mobility Score còn có thể đưa vào giá trị độ an toàn của Vua ở giai đoạn cuối trận cờ, với các nước mà quân Vua bị chiếu bí hoặc quân Vua bị hết nước đi.

Ta sẽ có hai giá trị của Vua bị chiếu bí và hết nước đi lần lượt là:

* Chiếu bí : -200000 (Tức là -200 Pawn).
* Hết nước đi: -150000 ( Tức là -150 Pawn).

**Mã giả**

***Function*** *evalMobility(length, depth)*

1. *Score = length \* 15 //Nhiều nước đi thì càng lợi thế.Được hiểu như Số nước đi có thể \* 75*
2. ***if*** *(length == 0) //Trường hợp chiếu bí hoặc không còn đường đi*
3. ***if*** *(!safeKing()) //Chiếu bí*
4. *Score = -200000 \* depth*
5. ***else*** *//Hết nước đi*
6. *Score = -150000 \* depth*
7. *Return Score*

**Hiện thực code**

**public static int** evalMobility(**int** length, **int** depth){  
 **int** score = 0;  
 score+= length \* 15; *// each move \* 5 \* 15* **if** (length == 0){ *//checkmate or stalemate* **if** (!*safeKing*()){  
 score-= 200000 \* depth;  
 }  
 **else**{  
 score-= 150000 \* depth;  
 }  
 }  
 **return** score;  
}

#### Đánh giá dựa trên độ tấn công của đối thủ (Attack Score)

Việc một quân cờ của bạn bị đối thủ tấn công, hăm dọa sẽ làm cho cơ hội chiến thắng của bạn giảm xuống. Thế nhưng có những việc phải đánh đổi quân cờ lại mang lại nhiều điều có lợi với những quân có giá trị cao hơn. Ý tưởng của phương pháp này rất đơn giản, với mỗi quân cờ đang bị đối thủ tấn công, thì điểm lợi thế của ta sẽ giảm. Để thực hiện được điều này, ta chỉ cần giả sử vị trị của quân cờ đó là quân Vua, xem quân đó có bị chiếu hay không (ta sử dụng lại cơ chế kiểm tra độ an toàn quân Vua).

Giá trị của Material Balance có thể tính đơn giản bởi công thức sau:

* i : là số thư tự biểu diễn loại cho quân cờ theo thứ tự của bảng trên. i=0 là Tốt, i=1 là Mã,….
* là gía trị số lượng quân cờ loại i màu Trắng đang bị đối thủ tấn công.
* là gía trị số lượng quân cờ loại i màu Đen đang bị đối thủ tấn công.
* là gía trị điểm của quân cờ loại i.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Tốt** | **Mã** | **Tượng** | **Xe** | **Hậu** |
| Giá trị | 50 | 350 | 400 | 525 | 1000 |

**Mã giả**

***Function*** *evalAttack()*

1. *tempKing = kingPositionU //Vị trí của quân Vua*
2. ***for*** *each Position*
3. *KingPositionU = Position //Giả sử quân cờ khác là quân Vua*
4. ***switch*** *Board[Position/8][Position%8]: do*
5. ***case*** *WhitePawn: do* ***if****(!safeKing()) Score-=ValuePawn //= “P”*
6. ***case*** *WhiteRook: do* ***if****(!safeKing()) Score-=ValueRook //= “R”*
7. ***case*** *WhiteKnight:do* ***if****(!safeKing()) Score-=ValueKnight //= “K”*

**Hiện thực code**

**public static int** evalAttack(){  
 *//who is under attacked* **int** score = 0;  
 **int** temp = *kingPositionU*;  
 **for**(**int** i = 0; i < 64; i++){  
 **switch** (*Board*[i/8][i%8]){  
 **case "P"** :  
 *kingPositionU* = i;  
 **if** (!*safeKing*()){ score-=50;}  
 **break**;  
 **case "K"** :  
 *kingPositionU* = i;  
 **if** (!*safeKing*()){ score-=350;}  
 **break**;  
 **case "B"** :  
 *kingPositionU* = i;  
 **if** (!*safeKing*()){ score-=400;}  
 **break**;  
 **case "R"** :  
 *kingPositionU* = i;  
 **if** (!*safeKing*()){ score-=525;}  
 **break**;  
 **case "Q"** :  
 *kingPositionU* = i;  
 **if** (!*safeKing*()){ score-=1000;}  
 **break**;  
 }  
 }  
 *kingPositionU* = temp;  
 **return** score;  
}

1. ***case*** *WhiteQueen: do* ***if****(!safeKing()) Score-=ValueQueen //=“Q”*
2. ***case*** *WhiteBishop:do* ***if****(!safeKing()) Score-=ValueBishop//=“B”*
3. *kingPositionU = tempKing //Trả lại vị trí quân Vua như ban đầu*
4. *Return Score*

### Xây dựng cơ chế thay đổi góc nhìn

Mong muốn giảm thiểu sự phức tạp trong cách xây dựng các nước đi của bất kỳ quân cờ nào bao gồm cả quân trắng và quân đen và trong việc xử lý các đánh giá, mà chỉ cần xây dựng các nước đi trên quân cờ trắng, đòi hỏi chúng ta phải có cách để chuyển đổi quân trắng thành quân đen và ngược lại.

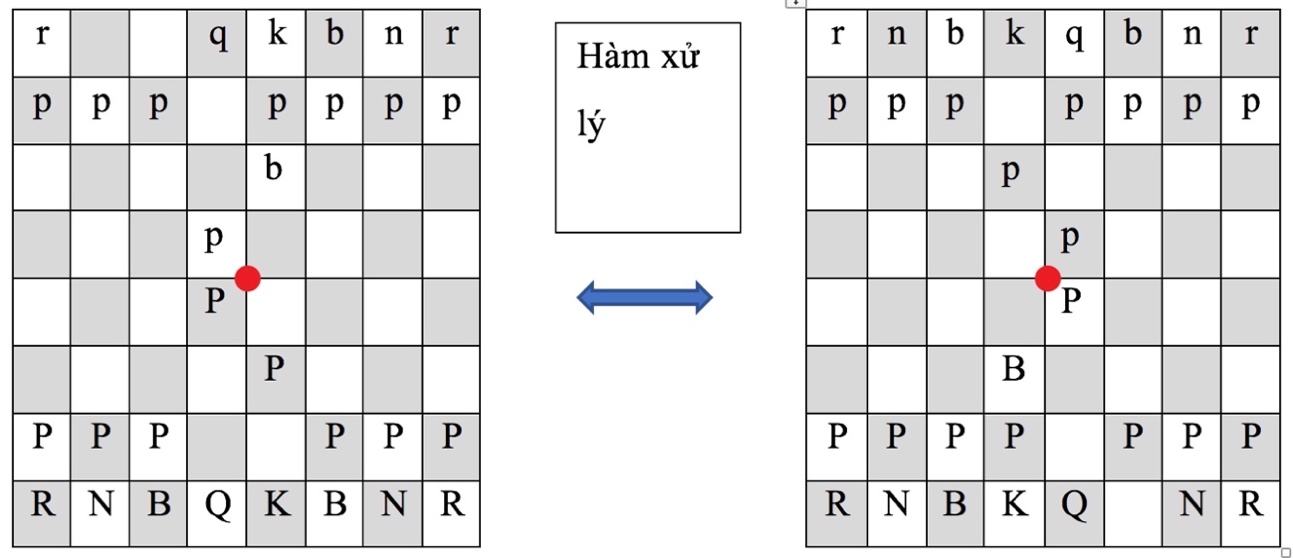
Hàm xử lý này sẽ thực hiện việc chuyển đổi với 3 nhiệm như sau:

* Tại mỗi ô vuông, sẽ thực hiện việc chuyển: nếu là giá trị đó in thường (quân đen) thì chuyển thành in hoa (quân trắng) hoặc trường hợp còn lại thì chuyển thành in thường(quân đen) (kể cả trường hợp ô vuông đó không bị chiếm giữ thì in thường vẫn giữ giá trị như vậy).
* Sẽ thực hiện việc trao đổi 2 giá trị giữa 2 ô vuông đang đối xứng nhau thông qua trọng tâm của bàn cờ hình vuông. VD: 2 ô vuông đối xứng nhau: [0][0] và [7][7] hoặc [2][4] và [5][3] . Vì bàn cờ là mảng 2 chiều 8x8 nên giới hạn giá trị là từ 0->7 đối với mỗi chiều. Nên 2 ô vuông được gọi là đối xứng nhau nếu có nó thõa mãn giá trị 2 chiều như sau: [Row][Col] và [7-Row][7-Col]. Vì thế khi hiện thực ta chỉ cần xét một nửa bàn cờ theo hàng ( từ 0->3 hoặc từ 4->7), từ một ô vuông ta có thể tìm được ô vuông đối xứng còn lại để trao đổi giá trị.
* Ngoài ra ta còn cập nhật lại biến chứa vị trí của Vua Trắng và vua Đen để có thể sử dụng biến này trong các hàm xử lý khác. Như đã giới thiệu ở phần trước, ta không cần dùng chỉ số hàng và cột nữa, ta sẽ quy đổi theo biến khác, biến này là một số nguyên có miền giá trị [0,63] tức là *Hàng \* 8 + Cột*. Ta sẽ tính công thức tính vị trí Vua Trắng như sau, Vua Đen cũng có cách tính ngược lại:

*Position\_KingWhiteNew = (7 - Row) \* 8 + 7 – Colume*

*= 63 – (Row \* 8 + Colume)*

*= 63 – Position\_KingBlackOld*



*Trọng tâm*

**Mã giả:**

***Function*** *Flip-Board()*

*// Swap - It’s similar with temp = a, a = b, b = temp.*

1. ***for*** *i = 0 -> 32* ***do:***
2. *Row <- i / 8* ***and*** *Col <- i % 8*
3. *Swap Board[row][col] with Board[7-row][7-col].*

*//in addition to if element is lower or upper then turn it into upper or lower.*

1. ***if*** *Board[Row][Col]* ***is*** *Lower*
2. *temp = Board[Row][Col] to Upper*
3. ***else*** *temp = Board[Row][Col] to Lower*
4. ***if*** *Board[7-Row][7-Col]* ***is*** *Lower*
5. *Board[Row][Col] = Board[7-Row][7-Col] to Upper*
6. ***else*** *Board[Row][Col] = Board[7-Row][7-Col] to Lower*
7. *Board[7-Row][7-Col] = temp*

*//We need to change the White’s king position*

1. *temp2 = kingPositionU*
2. *kingPositionU = 63 – kingPositionL*
3. *kingPositionL = temp2*

**Hiện thực code**

***public static void*** *flipBoard(){  
 String temp =* ***""****;* ***for****(****int*** *i = 0; i < 32; i++){* ***int*** *row = i/8, col = i % 8;* ***if****(Character.isLowerCase(Board[row][col].charAt(0))){  
 temp = Board[row][col].toUpperCase();  
 }* ***else****{  
 temp = Board[row][col].toLowerCase();  
 }* ***if****(Character.isLowerCase(Board[7 - row][7 - col].charAt(0))){  
 Board[row][col] = Board[7 - row][7 - col].toUpperCase();  
 }* ***else****{  
 Board[row][col]= Board[7 - row][7 - col].toLowerCase();  
 }  
 Board[7 - row][7 - col] = temp;  
 }* ***int*** *temp2 = 63 - kingPositionU;  
 kingPositionU = 63 - kingPositionL;  
 kingPositionL = temp2;  
}*

### Xây dựng cơ chế chuyển và phục hồi trạng thái

Một nước di chuyển của quân cờ được biễu diễn là một String có độ dài là 5 ký tự

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **String** | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** |
| Loại I | Hàng cũ | Cột cũ | Hàng mới | Cột mới | Quân cờ bị ăn |
| Loại II | Cột cũ | Cột mới | Quân cờ bị ăn | Quân cờ được phong | “P” |
| Loại III | Cột cũ của Vua | Cột cũ của Xe | Cột mới của Vua | Cột mới của Xe | “C” |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  | k. |  |  | |  |
| 3 |  | |  |  | | P |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | | K |  |  |  | |  |
| 4 |  | |  |  | |  |  | **.** |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

a) Mã đi từ (3,3) đến ô trống (4,5)

b) Tốt đi từ (3,3) đến ô (2,4) và ăn mã của đối thủ

**VD nước đi loại I:**

a) “3345 ”(chuỗi này có 5 ký tự, ký tự cuối cùng là khoảng trống)

b) “3324k”

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | | . |  |  |  | |  |
| 1 |  | |  |  | | P |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  | b. |  |  | |  |
| 1 |  | |  |  | | P |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  |  |  |  | |  |

d) Tốt đi từ (1,3) đến ô (0,4) và ăn tượng của đối thủ và phong hậu

c) Tốt đi từ (1,3) đến ô trống (0,3) và phong hậu

**VD nước đi loại II:** c) “33 QP” và d) “34bQP”

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 |  | |  |  | |  | A |  |  | | R |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | | 2 | 3 | | 4 | 5 | 6 | 7 | |
| 0 |  | |  |  | |  |  |  |  | |  |
| 1 |  | |  |  | |  |  |  |  | |  |
| 2 |  | |  |  | |  |  |  |  | |  |
| 3 |  | |  |  | |  |  |  |  | |  |
| 4 |  | |  |  | |  |  |  |  | |  |
| 5 |  | |  |  | |  |  |  |  | |  |
| 6 |  | |  |  | |  |  |  |  | |  |
| 7 | R | |  |  | |  | A |  |  | |  |

f) Vua ở ô (7,4) nhập thành xa với Xe ở ô (7,0) sẽ được Vua (7,2) và Xe (7,3)

e) Vua ở ô (7,4) nhập thành gần với Xe ở ô (7,7) sẽ được Vua (7,6) và Xe (7,5)

**VD nước đi loại III**: e) “4765C” và f) “4023C”

#### Thực hiện nước đi (ApplyMove):

Như trình bày trong phần Cấu trúc dữ liệu và mô tả không gian trạng thái. Hàm ApplyMove là một hàm thiết yếu và quan trọng:

* Tham số là một String – thể hiện nước đi thuộc lọai I, loại II hoặc loại III.
* Hàm không trả về giá trị. Múc đích của nó là thực hiện nước đi (tham số truyền vào) trên Board.

Vì tham số là nước đi thuộc loại I, loại II hoặc III (trang…) Mỗi loại có cấu trúc khác nhau, nên chúng ta sẽ xử lý theo từng trường hợp.

* Nếu ký tự thứ 5 trong chuỗi là ‘P’ sẽ mang ý nghĩa đây là nước đi loại II-Phong cấp cho tốt.

|  |  |  |
| --- | --- | --- |
|  | **Vị trí cũ** | **Vị trí mới** |
| x | 1 | 0 |
| y | String[0] | String[1] |
| Gía trị | blank | String[3] (Quân được phong) |

* Nếu ký tự thứ 5 trong chuỗi là ‘C’ sẽ mang ý nghĩa đây là nước đi loại III-Nhập thành cho Vua.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Vị trí Vua cũ** | **Vị trí Xe cũ** | **Vị trí Vua mới** | **Vị trí Xe mới** |
| x | 7 | 7 | 7 | 7 |
| y | String[0] | String[1] | String[2] | String[3] |
| Gía trị | blank | blank | A | R |

Cập nhật lại vị trí của vua *kingPositionU* = 7 \* 8 +String[2];

* Ngược lại thì là nước đi Loại I
  + Nếu quân cần khôi phục lại nước đi là quân Vua (String[4]=”A”) thì ta cần tính lại ví trí của Vua. Vua lúc này sẽ nằm ở vị trí cũ.  
     ***kingPositionU* = String[0] \* 8 +String[1];**

|  |  |  |
| --- | --- | --- |
|  | **Vị trí cũ** | **Vị trí mới** |
| x | String[0] | String[2] |
| y | String[1] | String[3] |
| Gía trị | blank | Quân ở vị trí cũ (đi đến vị trí mới) |

**Mã giả:**

*Function applyMove(Move) //String Move là chuỗi 5 ký tự*

*//Cấu trúc của từng loại I, II, III được giới thiệu rõ ở bảng trên.*

1. ***if*** *Move* ***is*** *type II //Khi Move[4] = “P”*
2. *Board[1][Move[0]] = “ ”*
3. *Board[0][Move[1]] = Move[3]*
4. ***else******if*** *Move* ***is*** *type III //Khi Move[4] = “C”*
5. *Board[7][Move[0]] = “ ”*
6. *Board[7][Move[1]] = “ ”*
7. *Board[7][Move[2]] = A*
8. *Board[7][Move[3]] = R*
9. *KingPositionU = 7 \* 8 + Move[2] //Cập nhật lại vị trí Vua*
10. ***else*** *//Loại I, các trường hợp còn lại*
11. *Board[Move[2]][Move[3]] = Board[Move[0])][Move[1]]*
12. *Board[Move[0]][Move[1]] = “ ” //Ô trống*
13. ***if*** *Board[Move[0]][Move[1]] = “A”*
14. *KingPositionU = Move[2] \* 8 + Move[3]*

**Hiện thực Code:**

***public static void*** *applyMove(String Move)  
{  
 // Return: list has a structure:  
 // normal: [row][col][next row][next col][piece at next row, next col]  
 // if pawn promotion : [col] [nex col] [captured piece] [promoted Piece] [P(pawn)]* ***if*** *(Move.charAt(4) ==* ***'P'****)  
 {  
 Board[1][Character.getNumericValue(Move.charAt(0))] =* ***" "****;  
 Board[0][Character.getNumericValue(Move.charAt(1))] = String.valueOf(Move.charAt(3));  
 }  
 //Colume [Previous King], [Rook ]=> [Next King], [Rook]* ***else if*** *(Move.charAt(4) ==* ***'C'****)  
 {  
 Board[7][Character.getNumericValue(Move.charAt(0))] =* ***" "****;  
 Board[7][Character.getNumericValue(Move.charAt(1))] =* ***" "****;  
 Board[7][Character.getNumericValue(Move.charAt(2))] =* ***"A"****;  
 Board[7][Character.getNumericValue(Move.charAt(3))] =* ***"R"****;  
 kingPositionU = 7 \* 8 + Character.getNumericValue(Move.charAt(2));  
 }  
 //[Previous Row, Col] [Next Row, Col] [Captured Piece]* ***else*** *{  
 Board[Character.getNumericValue(Move.charAt(2))][Character.getNumericValue(Move.charAt(3))] = Board[Character.getNumericValue(Move.charAt(0))][Character.getNumericValue(Move.charAt(1))];  
 Board[Character.getNumericValue(Move.charAt(0))][Character.getNumericValue(Move.charAt(1))] =* ***" "****;* ***if*** *(****"A"****.equals(Board[Character.getNumericValue(Move.charAt(2))][Character.getNumericValue(Move.charAt(3))]))  
 {  
 kingPositionU = Character.getNumericValue(Move.charAt(2)) \* 8 + Character.getNumericValue(Move.charAt(3));  
 }  
 }  
}*

#### Phục hồi nước đi (UndoMove):

Như trình bày trong phần Cấu trúc dữ liệu và mô tả không gian trạng thái. Hàm UndoMove là một hàm thiết yếu và quan trọng:

* Tham số là một String – thể hiện nước đi thuộc lọai I, loại II hoặc loại III.
* Hàm không trả về giá trị. Múc đích của nó là khôi phục lại nước đi (tham số truyền vào) trên Board.

Vì tham số là nước đi thuộc loại I, loại II hoặc III (trang…) Mỗi loại có cấu trúc khác nhau, nên chúng ta sẽ xử lý theo từng trường hợp.

* Nếu ký tự thứ 5 trong chuỗi là ‘P’ sẽ mang ý nghĩa đây là nước đi loại II-Phong cấp cho tốt.

|  |  |  |
| --- | --- | --- |
|  | **Vị trí cũ** | **Vị trí mới** |
| x | 1 | 0 |
| y | String[0] | String[1] |
| Gía trị | P | String[2] |

* Nếu ký tự thứ 5 trong chuỗi là ‘C’ sẽ mang ý nghĩa đây là nước đi loại III-Nhập thành cho Vua.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Vị trí Vua cũ** | **Vị trí Xe cũ** | **Vị trí Vua mới** | **Vị trí Xe mới** |
| x | 7 | 7 | 7 | 7 |
| y | String[0] | String[1] | String[2] | String[3] |
| Gía trị | P | R | blank | blank |

Cập nhật lại vị trí của vua kingPositionU = 7 \* 8 +String[0];

* Ngược lại thì là nước đi Loại I
  + Nếu quân cần khôi phục lại nước đi là quân Vua (String[4]=”A”) thì ta cần tính lại ví trí của Vua. Vua lúc này sẽ nằm ở vị trí cũ.

***kingPositionU = String[0] \* 8 +String[1];***

|  |  |  |
| --- | --- | --- |
|  | **Vị trí cũ** | **Vị trí mới** |
| x | String[0] | String[2] |
| y | String[1] | String[3] |
| Gía trị | Quân ở vị trí mới | String[4] |

**Mã giả:**

***Function*** *undoMove(Move) //String Move là chuỗi 5 ký tự*

*//Cấu trúc của từng loại I, II, III được giới thiệu rõ ở bảng trên.*

1. ***if*** *Move* ***is*** *type II //Khi Move[4] = “P”*
2. *Board[0][Move[0]] = P*
3. *Board[0][Move[1]] = Move[2]*
4. ***else******if*** *Move* ***is*** *type III //Khi Move[4] = “C”*
5. *Board[7][Move[0]] = A*
6. *Board[7][Move[1]] = R*
7. *Board[7][Move[2]] = “ ” //Ô trống*
8. *Board[7][Move[3]] “ ” //Ô trống*
9. *KingPositionU = 7 \* 8 + Move[0] //Cập nhật lại vị trí Vua*
10. ***else*** *//Loại I – các trường hợp còn lại*
11. *Board[Move[0]][Move[1]] = Board[Move[2]][Move[3]]*
12. *Board[Move[2]][Move[3]] = Move[4]*
13. ***if*** *Board[Move[0]][Move[1]] = “A”*
14. *KingPositionU = Move[0] \* 8 + Move[1]*

**Hiện thực Code:**

***public static void*** *undoMove(String Move)  
{  
 // Return: list has a structure:  
 // normal: [row][col][next row][next col][piece at next row, next col]  
 // if pawn promotion : [col] [nex col] [captured piece] [promoted Piece] [P(pawn)]* ***if*** *(Move.charAt(4) ==* ***'P'****)  
 {  
 Board[0][Character.getNumericValue(Move.charAt(1))] = String.valueOf(Move.charAt(2));  
 Board[1][Character.getNumericValue(Move.charAt(0))] =* ***"P"****;  
 }  
 //Colume [Previous King], [Rook ]=> [Next King], [Rook]* ***else if*** *(Move.charAt(4) ==* ***'C'****)  
 {  
 Board[7][Character.getNumericValue(Move.charAt(0))] =* ***"A"****;  
 Board[7][Character.getNumericValue(Move.charAt(1))] =* ***"R"****;  
 Board[7][Character.getNumericValue(Move.charAt(2))] =* ***" "****;  
 Board[7][Character.getNumericValue(Move.charAt(3))] =* ***" "****;  
 kingPositionU = 7 \* 8 + Character.getNumericValue(Move.charAt(0));  
 }  
 //[Previous Row, Col] [Next Row, Col] [Captured Piece]* ***else*** *{  
 Board[Character.getNumericValue(Move.charAt(0))][Character.getNumericValue(Move.charAt(1))] = Board[Character.getNumericValue(Move.charAt(2))][Character.getNumericValue(Move.charAt(3))];  
 Board[Character.getNumericValue(Move.charAt(2))][Character.getNumericValue(Move.charAt(3))] = String.valueOf(Move.charAt(4));* ***if*** *(****"A"****.equals(Board[Character.getNumericValue(Move.charAt(0))][Character.getNumericValue(Move.charAt(1))]))  
 {  
 kingPositionU = Character.getNumericValue(Move.charAt(0)) \* 8 + Character.getNumericValue(Move.charAt(1));  
 }  
 }  
}*

# CHƯƠNG 4: ỨNG DỤNG

## Giới thiệu chương trình ứng dụng

### Yêu cầu và chức năng của chương trình

Chương trình được cài đặt bằng ngôn ngữ Java trên công cụ hỗ trợ là Intellj IDE CE (IDE này được hỗ trợ đa nền tảng).

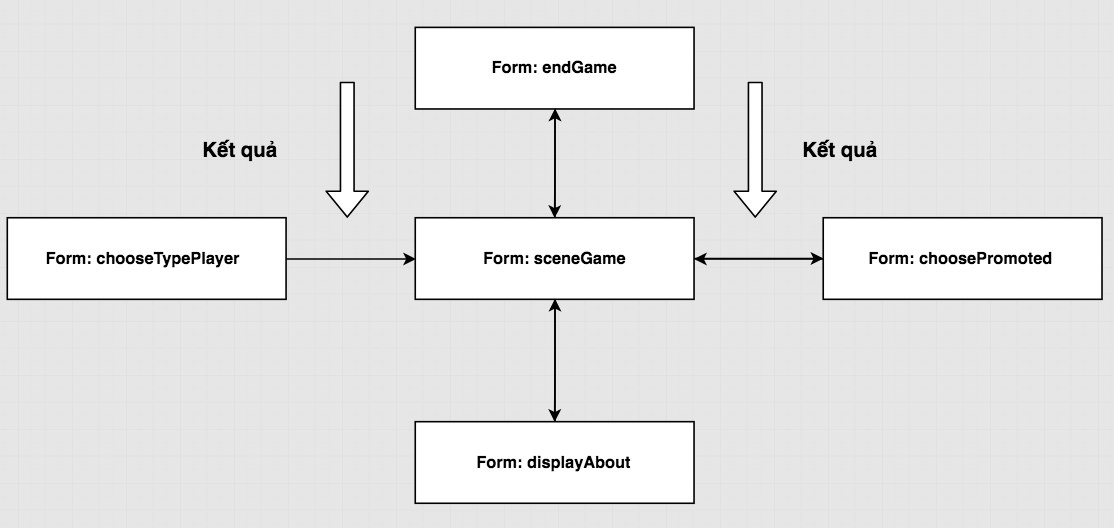
Chương trình gồm 2 chế độ

* Máy tính đánh trước.
* Con người đánh trước.

### Thiết kế giao diện chương trình

Giao diện chương trình được viết trên công nghệ JavaFX – một giải pháp công nghệ cho GUI trên nền tảng Java, nhằm tạo giao diện đồ họa cho người dùng một cách phong phú và mới mẻ hơn.

#### 4.1.2.1. Sơ đồ liên kết các màn hình



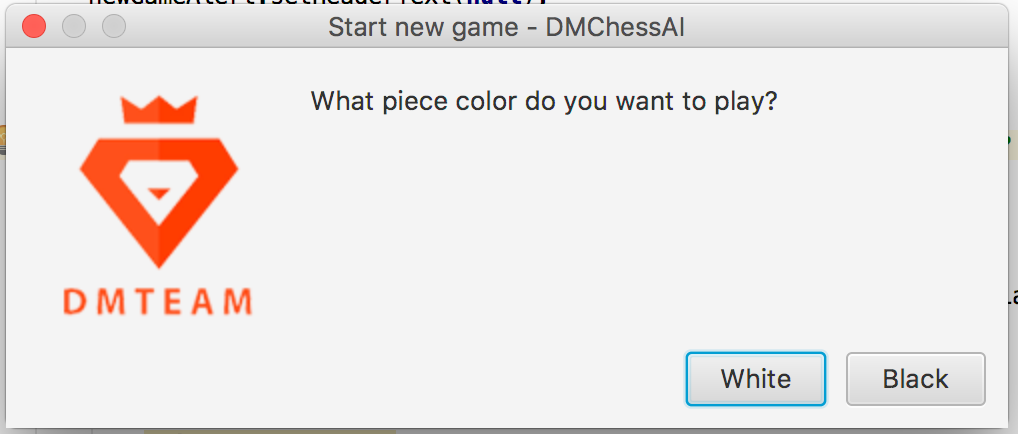
#### 4.1.2.2. Đặc tả giao diện

(Tên màn hình, chức năng; Các đối tượng trên màn hình: tên, kiểu, ràng buộc, chức năng).

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Màn hình** | **Loại màn hình** | **Chức năng** |
| 1 | chooseTypePlayer | Màn hình nhập liệu | Cho phép người dùng chọn màu quân cờ để đi trước hoặc đi sau. |
| 2 | sceneGame | Màn hình chính | Cho phép người dùng thực hiện các thao tác chơi trò chơi. Đồng thời các thao tác đến các màn hình khác. |
| 3 | displayAbout | Màn hình thông báo | Hiển thỉ thông báo về thông tin của chương trình. |
| 4 | choosePromoted | Màn hình nhập liệu | Cho phép người dùng chọn quân cờ mong muốn phong Tốt. |
| 5 | endGame | Màn hình thông báo | Hiển thị kết quả của trò chơi khi kết thúc. |

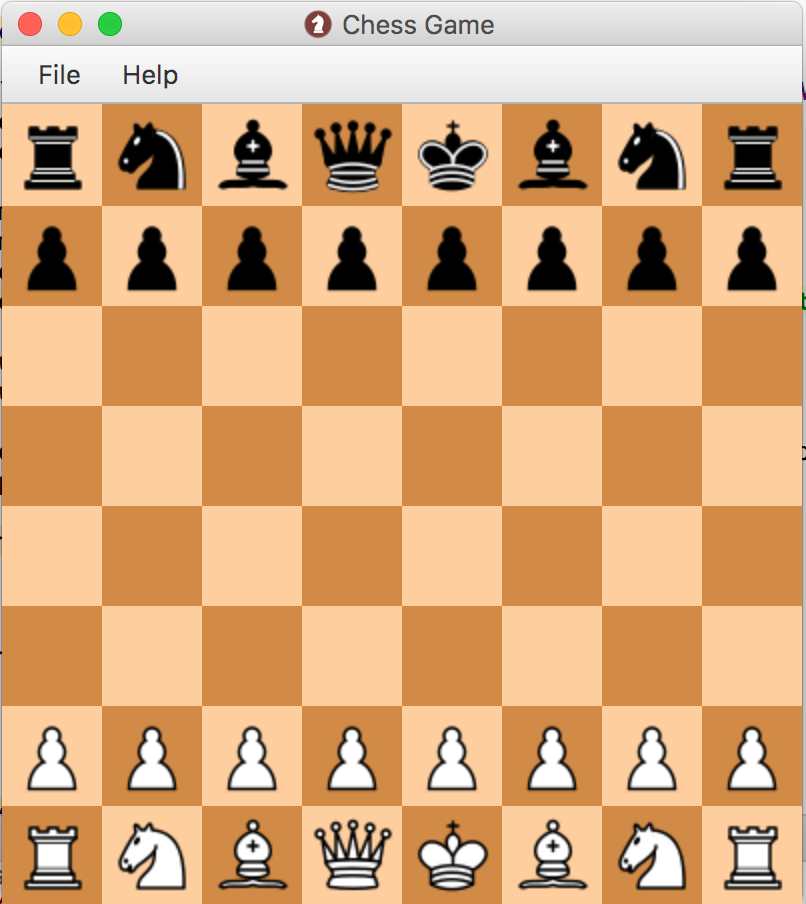
#### 4.1.2.3. Mô tả chi tiết các màn hình

**a) Giao diện Menu để chọn chế độ bắt đầu:**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | buttonTypeWhite | Button | Không có | Mở sceneGame với chế độ người chơi đi quân trắng trước. |
| 2 | buttonTypeBlack | Button | Không có | Mở sceneGame với chế độ người đi quân đen sau. |

**b) Giao** **diện GAME:**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | gameMenu | MenuBar | Không có | Giúp người chơi thao tác đến các màn hình khác |

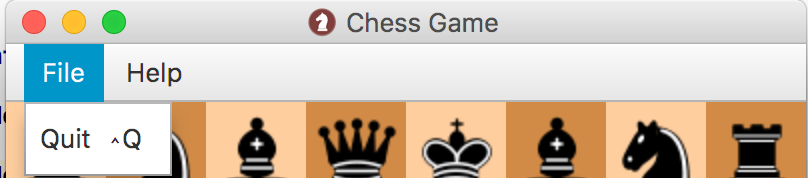
****

**Hướng dẫn chơi:**

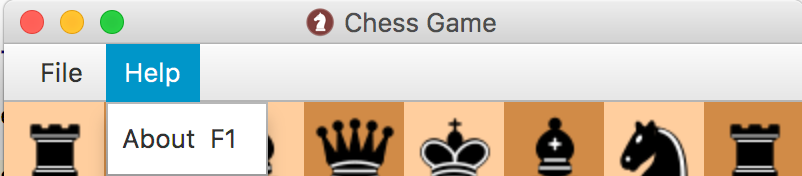
Tiến hành nhấp chuột vào vị trí quân cờ của mình, một hướng dẫn thực hiện nước đi sẽ hiện ra trên giao diện trò chơi.

* Sau đó nhấp chuột vào ô muốn đi tới.
* Để hủy, nhấp chuột vào vị trí quân cờ ban đầu

**c) Giao diện thanh Menu**

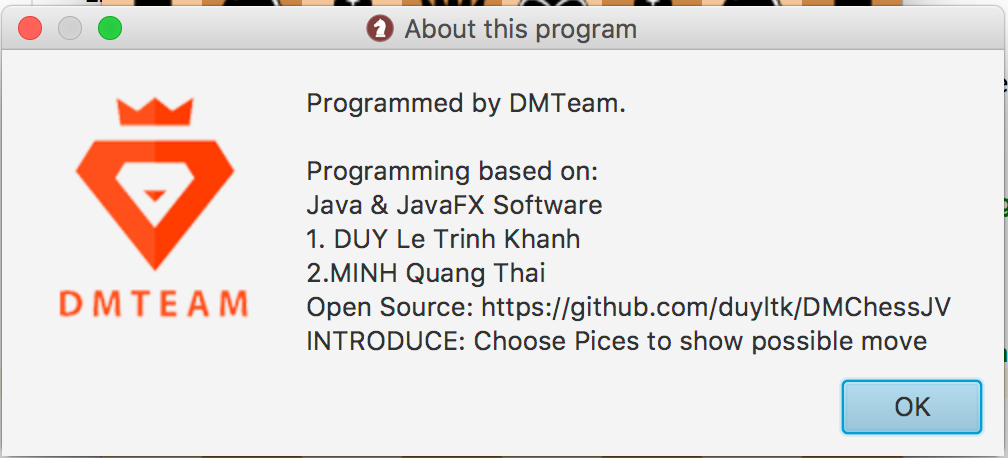
****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | menuItemQuit | Button | Không có | Giúp người chơi thoát Game. |



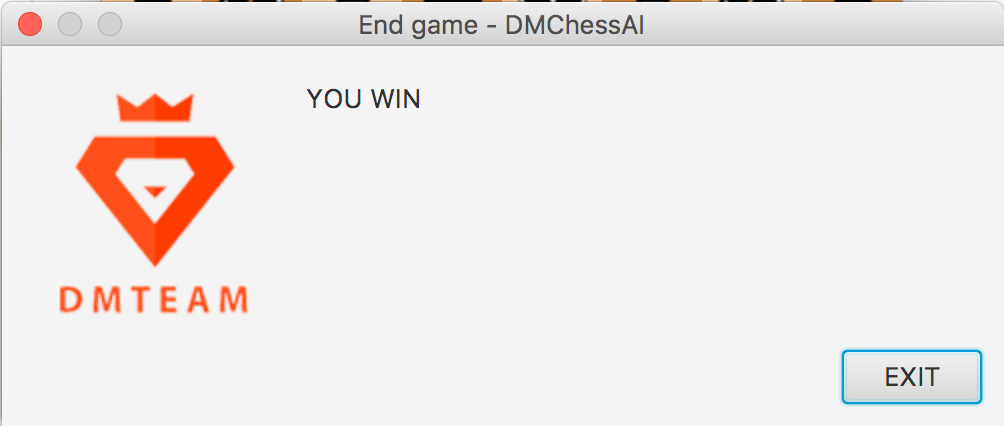
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | menuItemAbout | Button | Không có | Giúp người chơi chuyển đến màn hình thông tin chương trình |

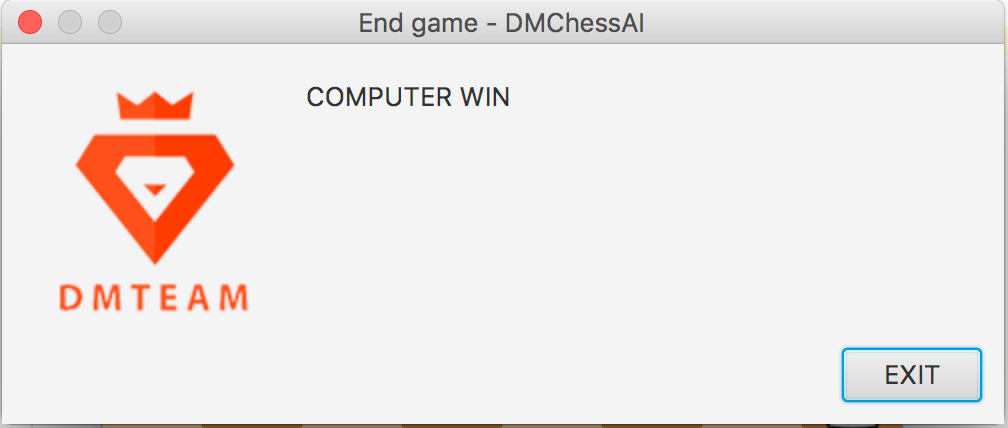
**d) Giao diện giới thiệu chương trình:**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | btnOK | Button | Không có | Thoát khỏi màn hình thông tin. Chuyển đến màn hình Scene Game. |

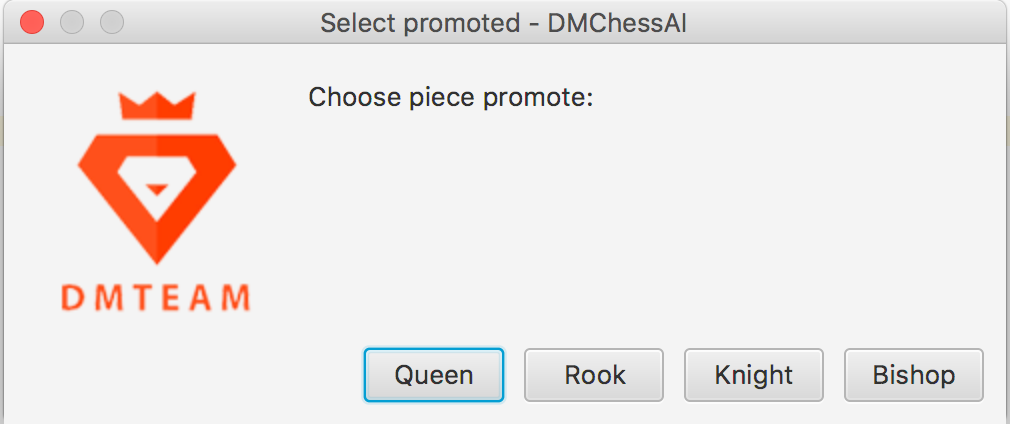
**e) Giao diện kết thúc trò chơi khi máy tính hoặc người chơi thắng:**

****

****

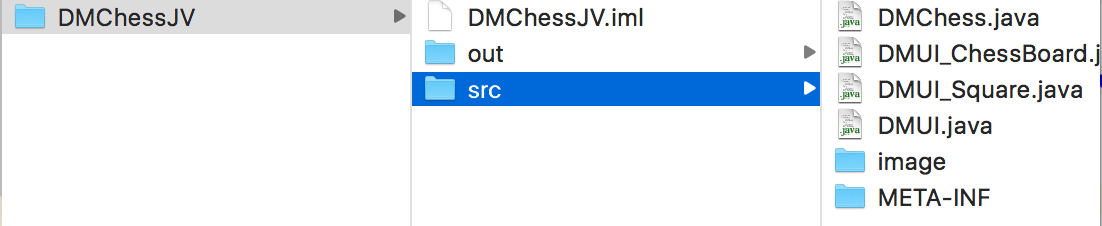
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | btnExit | Button | Không có | Thoát khỏi màn hình kết quả game. Chuyển đến màn hình Scene Game. |

**f) Màn hình phong cấp cho Tốt**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Kiểu** | **Ràng buộc** | **Chức năng** |
| 1 | buttonTypeQueen | Button | Không có | Phong Hậu cho quân Tốt |
| 2 | buttonTypeRook | Button | Không có | Phong Xe cho quân Tốt |
| 3 | buttonTypeKnight | Button | Không có | Phong Mã cho quân Tốt |
| 4 | buttonTypeBishop | Button | Không có | Phong Tượng cho quân Tốt |

### Hướng dẫn sử dụng chương trình



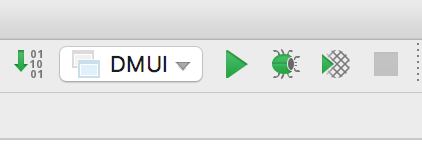
Bước 1: Ta có thư mục DMChessJV là thư mục chứa Project của bài toán.

Bước 2: Cần sử dụng IntellJ IDEA để mở được Project

* Click vào Open Project trên IntellJ IDEA. Và dẫn đường link về thư mục chứa Project.



* Sau khi đã có Project, tiến hành Build (Mũi tên đi xuống) và sau đó Run Project (Nút Play) để chương trình được thực thi.

****

## Cài đặt

### 4.2.1. Ngôn ngữ và công cụ sử dụng

**Ngôn ngữ**

Ngôn ngữ mà nhóm chọn để làm ra chương trình là ngôn ngữ Java. Java là một ngôn ngữ lập trình máy tính bậc cao, Java còn là một Platform (nền tảng). Khi Java là 1 ngôn ngữ lập trình bậc cao thì nó có nhiều tính năng như là: hướng đối tượng (Object Oriented), độc lập về nền tảng, đơn giản, bảo mật, mạnh mẽ, linh động …

Khác với phần lớn ngôn ngữ lập trình thông thường, thay vì biên dịch mã nguồn thành mã máy hoặc thông dịch mã nguồn khi chạy, Java được thiết kế để biên dịch mã nguồn thành bytecode, bytecode sau đó sẽ được môi trường thực thi (runtime environment) chạy.

Trước đây, Java chạy chậm hơn những ngôn ngữ dịch thẳng ra mã máy như C và C++, nhưng sau này nhờ công nghệ "biên dịch tại chỗ" - Just in time compilation, khoảng cách này đã được thu hẹp, và trong một số trường hợp đặc biệt Java có thể chạy nhanh hơn. Java chạy nhanh hơn những ngôn ngữ thông dịch như Python, Perl, PHP gấp nhiều lần. Java chạy tương đương so với C#, một ngôn ngữ khá tương đồng về mặt cú pháp và quá trình dịch/chạy.

Java ban đầu được phát triển bởi 1 nhóm có tên là Green Team thuộc công ty Sun Microsystems.

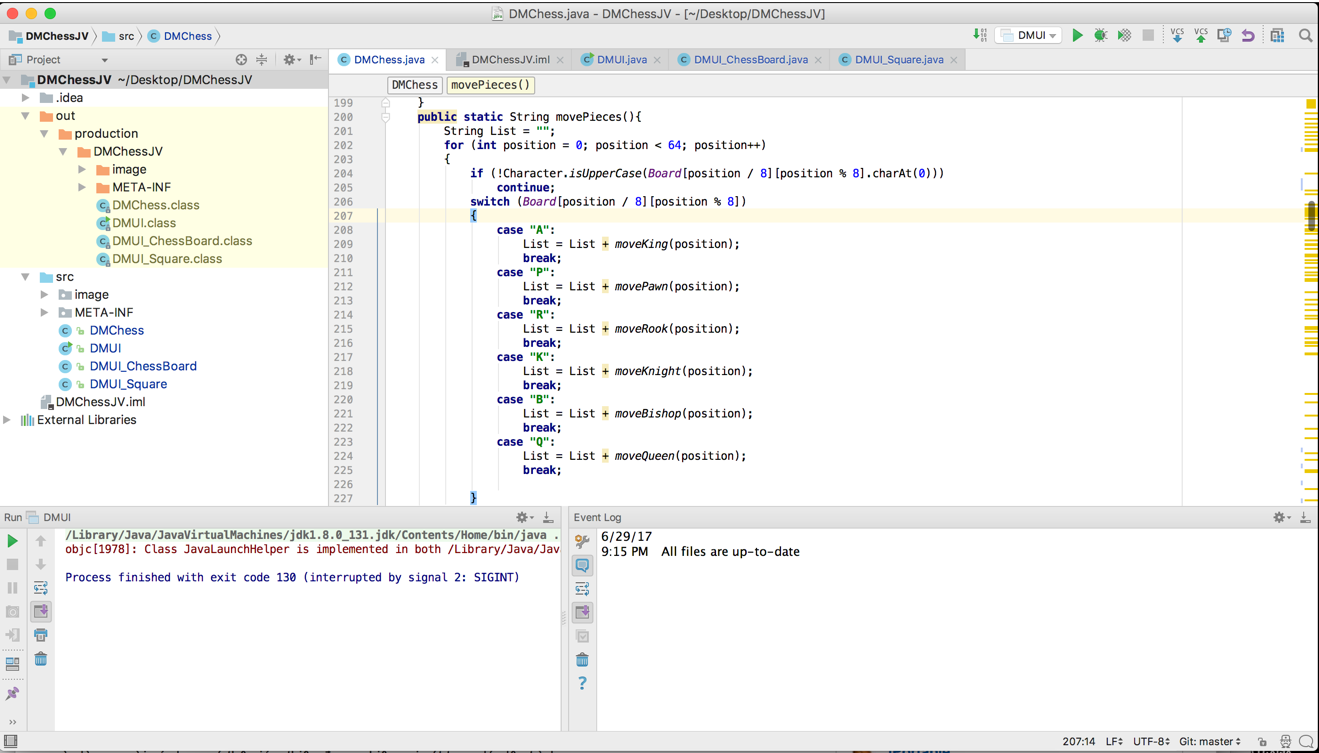
* Green Team bao gồm các thành viên James Gosling, Mike Sheridan, and Patrick Naughton. Nhóm này đã thành lập dự án về ngôn ngữ Java vào tháng 6 năm 1991.
* Ban đầu dự án có tên là “Green Talk” được đặt bời James Gosling.
* Sau đó nó được chuyển thành tên là : Oak (cây sồi), nguyên nhân có tên là cây sồi vì đây là biểu tượng cây xanh của 1 số quốc gia như Mỹ, Pháp, Đức …

Năm 1995, phiên bản Java đầu tiên được tung ra với phiên bản JDK 1.0 và từ đây cái tên Oak không còn được sử dụng nữa mà nó đổi hành Java tồn tại cho tới ngày hôm nay.

**Công cụ sử dụng**

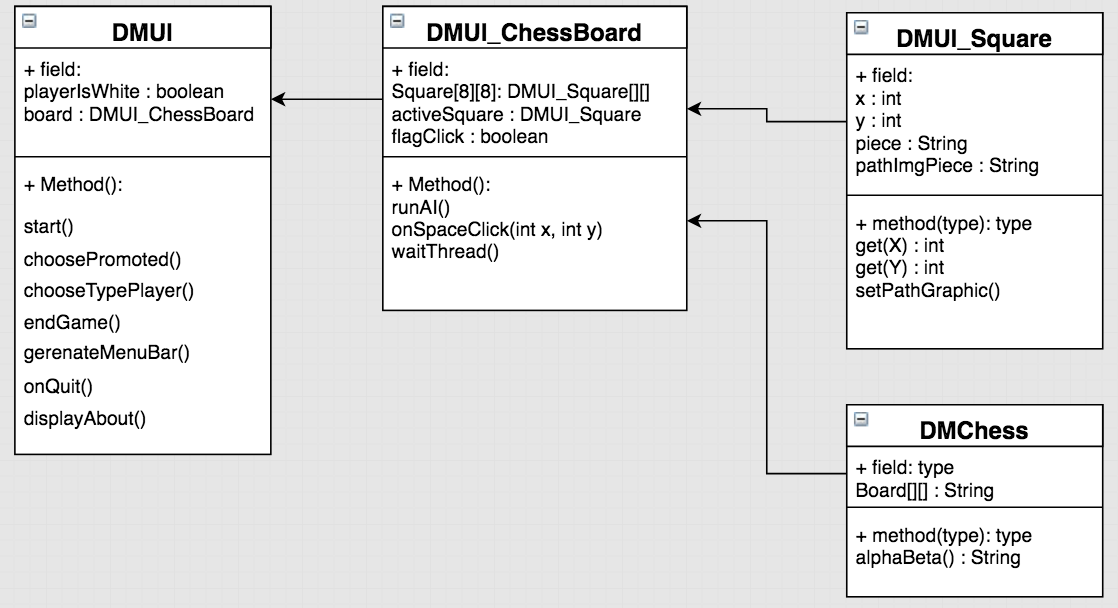
IntelliJ IDEA là phần mềm lập trình được phát triển JetBrains, nó cho phép phát triển các ngôn ngữ lập trình các ứng dụng cho điện thoại, Java, hoặc trên máy tính với việc hỗ trợ nhiều ngôn ngữ lập trình khác nhau phần mềm là một trong những giải pháp, chìa khóa trao tay dành cho những kỹ sư lập trình nhằm tạo ra những chương trình, ứng dụng cho máy tính, điện thoại hoặc bất kỳ thứ gì khác.

Giao diện chích của IntellJ IDEA:



### 4.2.2. Các hàm và đoạn code chính của chương trình

**Sơ đồ các lớp và các hàm sử dụng trong chương trình**

****

1. **Mô tả các Hàm của bảng DMUI**

|  |  |
| --- | --- |
| **Tên hàm** | **Vai trò** |
| start() | Khởi động cho chạy chương trình . |
| choosePromoted() | Khởi động màn hình hiển thị chọn quân phong cấp cho Tốt . |
| chooseTypePlayer() | Khởi động màn hình hiển thị chọn màu quân cờ của người chơi. |
| endGame() | Khởi động màn hình hiển thị kết quả của trò chơi. |
| generateMenuBar() | Khởi động dãy thanh tùy chọn trong màn hình Scene Game. |
| onQuit() | Tắt trò chơi. |

1. **Mô tả các Hàm của bảng DMUI\_ChessBoard**

|  |  |
| --- | --- |
| **Tên hàm** | **Vai trò** |
| runAI() | Khởi động truy xuất AI khi đến nước đi của máy tính cho trò chơi. |
| onSpaceClick() | Xử lý sự kiện click cho Scene Game. |
| waitThread() | Bởi vì JavaFX chạy song song với các lệnh thực thi, nên phải đợi render hình ảnh xong rồi mới thực thi các câu lệnh truy xuất AI. |

1. **Mô tả các Hàm của bảng DMUI\_Square**

|  |  |
| --- | --- |
| **Tên hàm** | **Vai trò** |
| getX() | Lấy giá trị tọa độ X tại ma trận game. |
| getY() | Lấy giá trị tọa độ Y tại ma trận game. |
| setPathGraphic() | Tiến hành truy xuất CSS để vẽ giao diện, đồng thời truy xuất các hình ảnh của quân cờ tại vị trí [X][Y] |

1. **Mô tả các Hàm của bảng DMChess**

Đã giởi thiệu và phân tích cụ thể ở mục 3.3.2.

# CHƯƠNG 5: KẾT LUẬN

## 5.1. Kết quả đạt được

Chương trình hoàn thiện được các yêu cầu cơ bản mà đồ án đặt ra: Cho phép máy tính đánh với người.

AI có khả năng chơi cờ vua ngang ngửa như một người chơi trung bình trong một CLB, với mức suy nghĩ tầm 6s, và khá đối khi mức suy nghĩ gần 16s.

Giải quyết bài toán theo đúng mục tiêu theo kiến thức đã học và nghiên cứu thêm những kiến thức mơi.

## 5.2. Hạn chế

Giao diện còn đơn giản.

Tốc độ chương trình chưa tối ưu do không gian trạng thái của bài toán lớn, để mở được đến trạng thái đích chương trình phải xét qua rất nhiều trạng thái trung gian.

Khả năng đánh giá còn sơ khai, cần được cải thiện nhiều hơn.

## 5.3. Hướng phát triển

Hoàn thiện giao diện.

Nghiên cứu thêm các phương pháp đánh giá để giảm thiểu không gian trạng thái cho thời gian thực thi tốt hơn.

Thực hiện nghiên cứu thêm nhiều hàm đánh giá phức tạp để máy tính trở nên thông minh hơn

# TÀI LIỆU THAM KHẢO

[1] Wikipedia, *History of chess*. *https://en.wikipedia.org/wiki/Chess#History*

[2] Wikipedia, *Chess*. *https://en.wikipedia.org/wiki/Chess*

[3] Chessbase.com (2012), *AGON releases new chess player statistics from YouGov*. *http://en.chessbase.com/post/che-redux-how-many-people-play-che-*

[4] Wikipedia, *Movement of chess*. *https://en.wikipedia.org/wiki/Chess#Movement*

[5] Wikipedia, *Deep Blue (Chess computer)*. *https://en.wikipedia.org/wiki/Deep\_Blue\_(chess\_computer)*

[6] Wikipedia, *Minimax*. *https://en.wikipedia.org/wiki/Minimax*

[7] Stdio.vn (2015), *Giải thuật tìm kiếm Minimax*.

*https://www.stdio.vn/articles/read/283/giai-thuat-tim-kiem-minimax*

[8] Stdio.vn (2017), *Giải thuật cắt tỉa Alpha-Beta*.

*https://www.stdio.vn/articles/read/564/giai-thuat-cat-tia-alpha-beta*

[9] Wikipedia, *Alpha-Beta Prunning*.

*https://en.wikipedia.org/wiki/Alpha%E2%80%93beta\_pruning*

[10] COMP9414/9814 Artificial Intelligence (2014), *Solutions to Alpha-Beta Exercises.*

*http://www.cse.unsw.edu.au/~cs9414/15s1/tut/sol/wk04sol.html*

[11] Wikipedia, *Claude Shannon*. *https://en.wikipedia.org/wiki/Claude\_Shannon*

[12] Claude Shannon (1949), Bài báo khoa học: *XXII. Programming a Computer for Playing Chess.*

[13] Chess Programming Wiki, *Evaluation*. https://chessprogramming.wikispaces.com/Evaluation

[14] Wikipedia, *Rook (Chess). https://en.wikipedia.org/wiki/Rook\_(chess)*

[15] Wikipedia, *Bishop (Chess). https://en.wikipedia.org/wiki/Bishop\_(chess)*

[16] Wikipedia, *Knight (Chess). https://en.wikipedia.org/wiki/Knight\_(chess)*

[17] Wikipedia, *Queen (Chess). https://en.wikipedia.org/wiki/Queen\_(chess)*

[18] Wikipedia, *Pawn (Chess). https://en.wikipedia.org/wiki/Pawn\_(chess)*

[19] Wikipedia, *King (Chess).* *https://en.wikipedia.org/wiki/King \_(chess)*

[20] Chessbase.com (2011), *The joys of chess – and the value of the pieces.* *http://en.chessbase.com/post/the-joys-of-che-and-the-value-of-the-pieces*

[21] Chess Programming Wiki*, Point value.*

*https://chessprogramming.wikispaces.com/Point+Value*

[22] Chess Programming Wiki, *Piece-Square Tables.*

*https://chessprogramming.wikispaces.com/Piece-Square+Tables*

[23] Sacha Droste, Johannes Fürnkranz (2008), *Learning of Pieces of Values for Chess Variants.*

*http://www.ke.tu-darmstadt.de/publications/reports/tud-ke-2008-07.pdf*

[24] Eliot Slater (1988), *Statistics for the Chess Computer and the Factor of Mobility.*

*http://eliotslater.org/index.php/chess/147-statistics-for-the-chess-computer-and-the-factor-of-mobility*

[25] Chess Programming Wiki, *Minimax*.

https://chessprogramming.wikispaces.com/Minimax

[26] Lauri Hartikka, *A step-by-step guide to building a simple chess AI.* *https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977*

[27] Chess Programming Wiki, *Alpha-Beta. https://chessprogramming.wikispaces.com/Alpha-Beta*

# PHỤ LỤC

**HỢP ĐỒNG NHÓM**

1. **Tên nhóm :** DMTeam.
2. **Thời gian và địa điểm :**

* Thứ 5, ngày 1 tháng 6 năm 2017.
* Địa điểm : KTX khu A, ĐHQG.

1. **Thành viên nhóm:**

|  |  |  |
| --- | --- | --- |
| **STT** | **MSSV** | **Họ và tên** |
| **1** | 15520159 | Lê Trịnh Khánh Duy |
| **2** | 15520494 | Thái Quang Minh |

1. **Mục đích thành lập nhóm:**

* Thực hiện đề tài môn : Trí tuệ nhân tạo (Trò chơi cờ vua).
* Rèn luyện kỹ năng lãnh đạo, kỹ năng làm việc nhóm, kỹ năng quản lý thời gian, kỹ năng tư duy và sáng tạo.
* Nâng cao khả năng làm việc theo chuẩn, cùng nhau giúp đỡ để cùng nhau tiến bộ trong học tập và hoàn thiện những kỹ năng mà môn học yêu cầu.
* Kết thúc môn học nắm được : các kiến thức lịch sử hình thành, định hướng nghiên cứu khoa học và ứng dụng của Trí tuệ nhân tạo. Các phương pháp giải quyết vấn đề và áp dụng, chú trọng phương pháp heuristic. Một số phương pháp biểu diễn tri thức cơ bản và kĩ thuật xử lý tri thức.

1. **Phân công công việc**

|  |  |  |  |
| --- | --- | --- | --- |
| **Thành viên** | **Viết báo cáo** | **Thiết kế giao diện** | **Hiện thực thuật toán** |
| Lê Trịnh Khánh Duy | X | X | X |
| Thái Quang Minh | X | X | X |

1. **Chỉ tiêu đánh giá**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Đặc điểm** | **Rất tốt** | **Tốt** | **Bình thường** | **Chưa tốt** |
| Trách nhiệm | Hoàn thành tốt công việc được phân công,  nhiệt tình giúp đỡ nhau | Hoàn thành tốt các công việc được phân công | Hoàn thành công việc | Không hoàn thành công việc |
| Thái độ, hợp tác | Tôn trọng, giúp đỡ lẫn nhau, sẵn sàng nhận nhiệm vụ để hoàn thành tốt  công việc là lợi  ích của nhóm | Tôn trọng, giúp đỡ lẫn nhau hoàn thành công việc. | Hoàn thành công việc và giúp đỡ một phần công việc. | Không hoàn thành công việc  và ít giúp đỡ các thành viên khác. |
| Quản lý thời gian | Hoàn thành nhiệm vụ trước thời hạn và luôn đúng giờ  trong các buổi họp | Hoàn thành nhiệm vụ đúng hạn và trễ không quá 5  phút các buổi  họp nhóm | Hoàn thành nhiệm vụ với sự nhắc nhở và trễ không quá  15 phút có lý do trong các buổi họp nhóm | Không hoàn thành nhiệm vụ hoặc trễ quá 15 phút hoặc vắng  không lý do trong các buổi họp nhóm |
| Nêu ý kiến | Tích cực nêu ý kiến, tìm kiếm giải pháp để  giải quyết các vấn đề tốt hơn. | Nêu ý kiến khi có đề xuất giải pháp để giải quyết vấn đề | Chỉ nêu ý kiến khi có sự nhắc nhở | Không nêu ý kiến |
| Tìm kiếm thông tin | Tìm kiếm thông tin đầy  đủ, phong phú,  tổng hợp thông tin hiệu quả | Tìm kiếm thông tin đầy đủ, phong phú | Tìm kiếm thông tin đúng  với phần nhiệm vụ được giao | Không tìm kiếm, đưa thông tin kiểu đối phó |
| Giữ liên lạc | Luôn giữ liên lạc và họp nhóm đầy đủ | Họp nhóm đầy đủ. Liên lạc  dưới 2 lần mới được. | Họp nhóm đầy đủ khi có sự nhắc nhở. | Không liên lạc được và ít khi họp nhóm. |

1. **Hiệp thương**
   * Hiệp thương thông qua Hợp đồng nhóm.
   * Ra quyết định theo nguyên tắc đồng thuận.
   * Hoàn thành nhiệm vụ đúng thời hạn.
   * Tôn trọng lẫn nhau giữa các thành viên và một mục đích chung là hoàn thành xuất sắc đồ án môn học.
2. **Cam kết**
   * Sau khi hiệp thương thông qua Hợp đồng nhóm, các thành viên cam kết thực hiện đúng yêu cầu, nhiệm vụ đã được phân công.
   * Ký tên

|  |  |
| --- | --- |
| **Thành viên** | **Ký tên** |
| Thành viên 1  Lê Trịnh Khánh Duy |  |
| Thành viên 2  Thái Quang Minh |  |

*Tp, Hồ Chí Minh, ngày 01 tháng 6 năm 2017*

**ĐÁNH GIÁ NỘI BỘ**

|  |  |
| --- | --- |
| **Thành viên** | **Đánh giá** |
| Thành viên 1  Lê Trịnh Khánh Duy | 95% |
| Thành viên 2  Thái Quang Minh | 95% |

Các thành viên đều cố gắng hết sức trong quá trình làm đồ án để hoàn thành công việc được giao. Trong quá trình làm đồ án tuy gặp rất nhiều khó khăn nhưng các thành viên trong nhóm đều cùng nhau nghiên cứu và tìm hướng giải quyết.