ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

****

BÁO CÁO BÀI TẬP LỚN

Học phần: Nhập môn trí tuệ nhân tạo

Đề tài: Chess Bot

Giảng viên hướng dẫn: Trần Thế Hùng

Nhóm: 24

Thành viên: Đàm Trần Ngọc Đức - 20210208
Nguyễn Huy Hoàng - 20215056
Nguyễn Mạnh Hiếu - 20215050
Nguyễn Văn Quý - 20210729
Trần Trung Kiên - 20210494

Hà Nội, tháng 6 năm 2024

Mục lục

Phân chia công việc:

Chương I. Tổng quan về game

1.1. Giới thiệu về game

Game cờ vua một người chơi, đối thủ là bot chơi cờ vua tự động (sau đây gọi là Chess bot), có thể tự đưa ra các nước đi hợp lý tương đương với một người chơi cờ vua có chỉ số elo 1000 – 1500 (thang đo chỉ mang tính tương đối).

1.2. Ý tưởng

Cờ vua là một trò chơi dạng boardgame dành cho 2 người. Một trận đấu chỉ có 3 kết quả có thể xảy ra: A thắng B thua, A thua B thắng, A và B hòa nhau. Cờ vua có thể được giải ngay từ nước đi đầu tiên một cách tuyệt đối (xác định được A thắng hay B thắng) (định lý Zermelo), tuy nhiên ước tính số thế cờ hợp lệ trong cờ vua là 4x1044, với độ phức tạp vào khoảng 10123, quá lâu để có thể tính toán kể cả với những công nghệ hiện đại ngày nay. Tuy nhiên, có thể dùng định lý trên có thể sử dụng các thuật toán để tìm ra được nước đi đủ tốt ở một độ sâu không quá phức tạp .

1.3. GDD

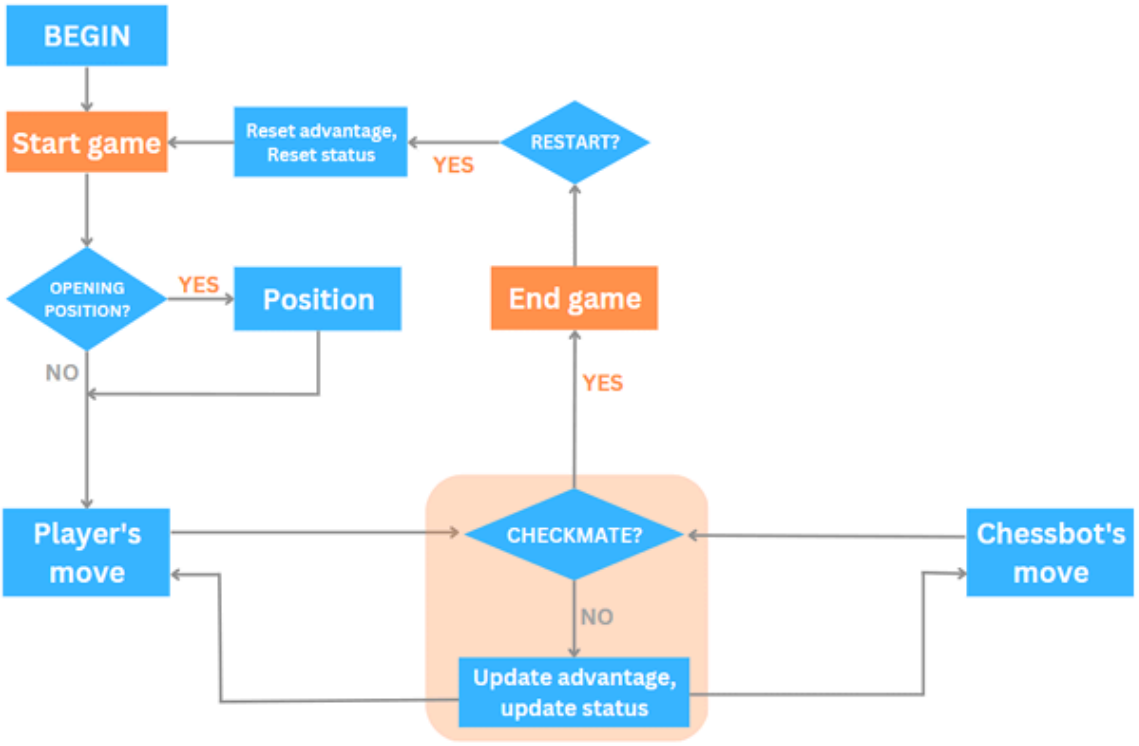
Gameplay

Bàn cờ vua 8x8 truyền thống và các quân cờ. Người di chuyển các quân cờ theo luật cờ vua, suy nghĩ sử dụng các chiến lược và chiến thuật để dành chiến thắng trước đối thủ (Chessbot).

Core loop

Từng bước giành chiến thắng bằng cách di chuyển các nước có lợi thế cho bản thân hoặc bất lợi cho đối thủ xuyên suốt cả ván đấu. Mỗi một nước đi có lợi, game sẽ thông báo số điểm lợi thế hiện tại trên giao diện, giúp người chơi dễ dàng theo dõi trạng thái hiện tại và suy nghĩ thế trận tiếp theo.

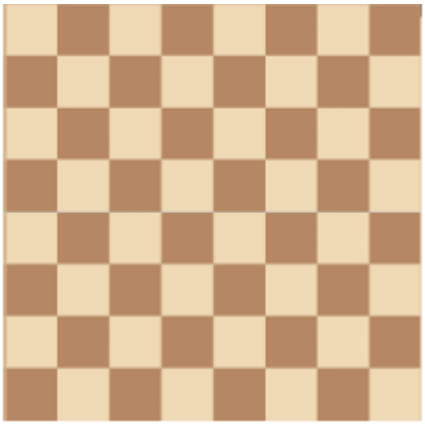
Game flow



Object

Các thành phần trong game được chia làm 2 phần chính: Bàn cờ và các thanh cài đặt.

Bàn cờ: Bàn cờ tiêu chuẩn 8x8



Các quân cờ vua (đen và trắng)


d
e
f
g
h

Chương II. Phân tích thiết kế Game

2.1. SAN và FEN

Ký hiệu Đại số ngắn (Short Algebraic Notation - SAN) là một phương pháp ghi chép các nước đi trong cờ vua một cách ngắn gọn bằng các ký hiệu và chữ cái tắt. SAN rất phổ biến trong cờ vua để ghi lại các trận đấu, trong đó mỗi nước đi được ghi bằng cách sử dụng các ký tự ngắn gọn để diễn tả quân cờ và ô cờ đích. Các ký tự được sử dụng trong SAN để diễn tả quân cờ như sau:

K: King (Vua)
Q: Queen (Hậu)
R: Rook (Xe)
B: Bishop (Tượng)
N: Knight (Mã)

Các ký tự tương ứng với ô cờ đích được sử dụng để diễn tả các nước đi: a, b, c, d, e, f, g, h: các cột trên bàn cờ (từ trái sang phải) 1, 2, 3, 4, 5, 6, 7, 8: các hàng trên bàn cờ (từ dưới lên trên). Tên nước đi đầy đủ có dạng: [Tên quân cờ][Ô xuất phát]-[Ô đích].
Ví dụ: Quân Trắng đi Hậu từ ô d1 đến ô d5 được viết là: Hd1-d5.
Nước ăn quân thường được ký hiệu bằng dấu "×". Ví dụ: Hd4×a4, đối với Tốt thì chỉ cần ghi ký hiệu hai cột trước và sau khi ăn, không có dấu nhân, chẳng hạn gf, bc. Tên của quân xuất hiện sau khi phong cấp được ghi sau nước đi, hoặc có thể dùng thêm dấu "/" hoặc "=" rồi đến tên quân phong cấp. VD: f8H, f8/H.
Nước nhập thành gần được ký hiệu là 0-0, còn nhập thành xa là 0-0-0. Nước chiếu vua có ký hiệu là dấu cộng (+), nước chiếu đôi có kí hiệu là hai dấu cộng (++) và chiếu hết có ký hiệu là dấu thăng (#).

Ký hiệu Forsyth–Edwards (FEN) là một ký hiệu tiêu chuẩn để mô tả một vị trí bàn cờ cụ thể của một ván cờ vua. Mục đích của FEN là cung cấp tất cả thông tin cần thiết để bắt đầu lại trò chơi từ một vị trí cụ thể.
Một chuỗi FEN bao gồm 6 phần, được ngăn cách nhau bằng 1 dấu cách:

1. Vị trí của các quân cờ từ bên trắng (phía dưới bàn cờ) đến bên đen (phía trên bàn cờ) được biểu diễn bởi một chuỗi ký tự. Các ký tự đại diện cho các quân cờ được trình bày ở phần “Danh mục các ký hiệu và chữ viết tắt”. Các ký tự được sắp xếp theo thứ tự từ a đến h, sau đó từ hàng 1 đến hàng 8. Mỗi hàng được ngăn cách bởi dấu /. Nếu trên một ô không có quân cờ, sử dụng một số lượng dấu cách tương ứng.
Với chuỗi FEN rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR, đây là vị trí bắt đầu của các quân cờ trên bàn cờ cờ vua.

2. Ký tự w hoặc b đại diện cho lượt đi của người chơi tiếp theo. w đại diện cho bên trắng và b đại diện cho bên đen.

3. Các ký tự K, Q, k và q đại diện cho khả năng castling của các bên: K cho biết bên trắng có thể castling với xe bên phải (khi chưa di chuyển). Q cho biết bên trắng có thể castling với xe bên trái (khi chưa di chuyển). k cho biết bên đen có thể castling với xe bên phải (khi chưa di chuyển). q cho biết bên đen có thể castling với xe bên trái (khi chưa di chuyển).

4. Kí tự đại diện cho vị trí En passant (có thể bỏ qua được) là một trong các quy tắc đặc biệt của cờ vua. Khi một quân tốt của bên đối phương di chuyển hai ô từ hàng ban đầu của nó và đi qua ô của người chơi, người chơi đối phương có thể bắt quân tốt đó như là một nước ăn En passant. Ký tự - đại diện cho trường hợp không có En passant.

5. Số hiệp số của trò chơi, là số nước đi đã thực hiện kể từ lần cuối cùng một quân tốt di chuyển hoặc một quân cờ bị ăn. Nó được tính bằng cách đếm số nước đi sau khi xảy ra hành động đó. Nếu không có hành động này xảy ra trong một nước đi, thì hiệp số được giữ nguyên từ nước đi trước đó. Hiệp số được sử dụng để xác định nếu một trận đấu có thể được xác định là hòa bởi luật 50 nước. Nếu không có quân tốt được di chuyển hoặc quân cờ bị ăn trong vòng 50 nước (100 bước đi của cả hai bên), thì trận đấu sẽ được xác định là hòa bởi luật 50 nước. Số này được cập nhật sau mỗi nước đi của bên đen. Mỗi hiệp số là một nước đi của bên trắng và một nước đi của bên đen.

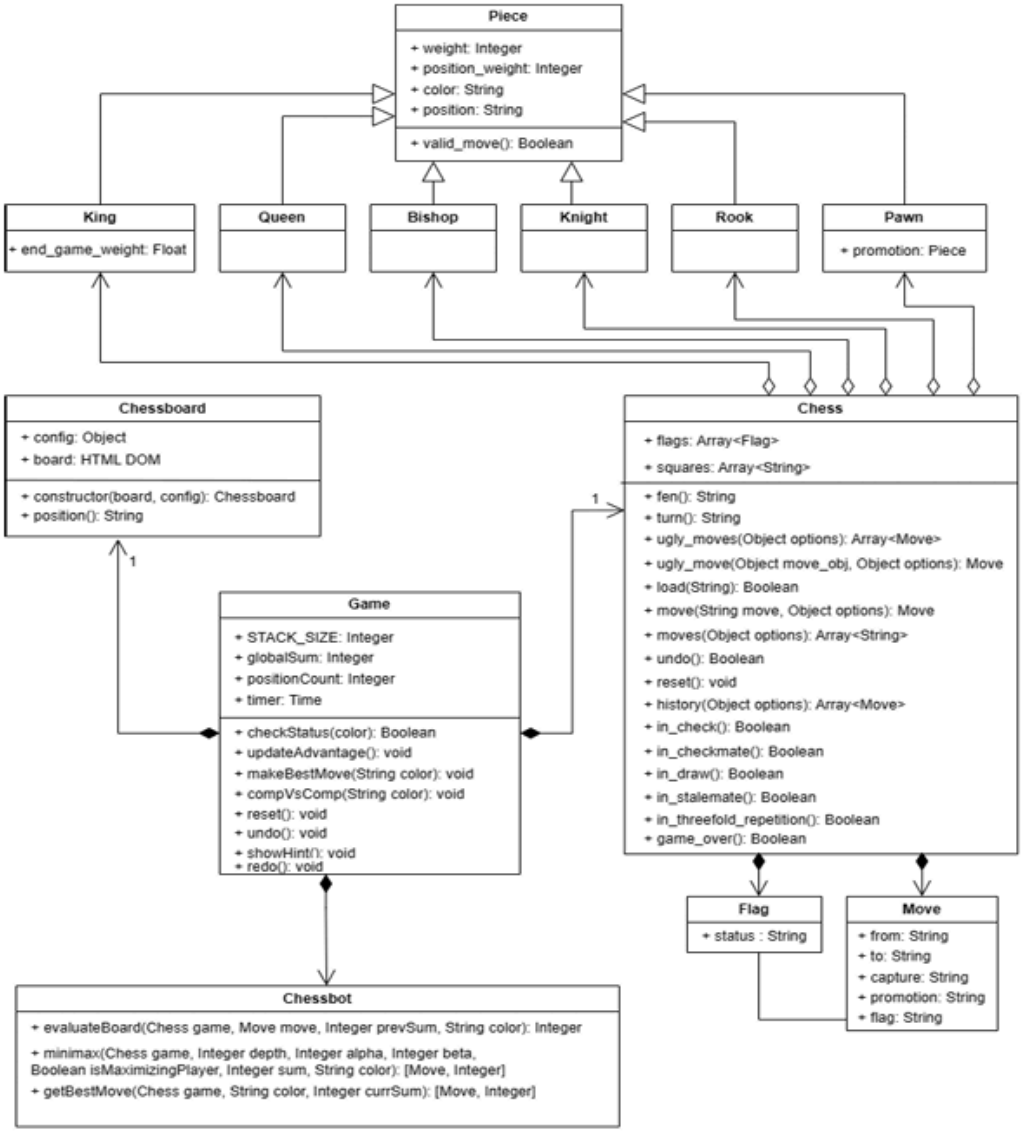
6. Số lượng nước đi của mỗi bên, được lưu trữ bằng một số nguyên. Nếu giá trị này là 0, nó có nghĩa là chưa có nước đi nào được thực hiện. Nếu giá trị này được cập nhật, nó sẽ tăng thêm một đơn vị sau mỗi lượt đi của bên đen.

Như vậy, “rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1” chính là chuỗi FEN bắt đầu một trận đấu trong cờ vua.

Ký hiệu SAN và FEN rất quan trọng. Nó giúp xác định được vị trí các quân cờ hiện tại trên bàn cờ, tình trạng của các quân cờ, trạng thái ván đấu và các nước đi đặc biệt. Hơn nữa, nó là các tham số truyền vào và kết quả trả về của các phương thức trong Chess.js và Chessboard.js, đồng thời từ chuỗi fen giúp Chessbot có thể biết được toàn bộ trạng thái trên bàn cờ hiện tại để tính toán và đưa ra nước đi hợp lý.

2.2. Sơ đồ lớp

- Game được xây dựng dựa trên các hàm chức năng là chính, vậy nên các đối tượng trong game sẽ rất tổng quát (ví dụ như bàn cờ là một đối tượng gồm các phương thức để tương tác với giao diện là chính, trong khi đó “điều khiển bàn cờ” lại là một đối tượng khác bao gồm các phương thức để điều khiển các quân cờ theo luật, cũng như kiểm tra trạng thái ván cờ hiện tại (chiếu, chiếu hết,…)). Chia đối tượng như vậy một phần là do các phương thức đã được tích hợp sẵn trong các thư viện Chessboard.js và Chess.js, hơn nữa chúng cũng thuận tiện cho việc triển khai mã nguồn và phát triển phần mềm theo hướng chức năng.



2.3. Cơ sở lý thuyết xây dựng Chess bot

- Ý tưởng đưa ra nước đi tối ưu:

Trong lý thuyết trò chơi, trò chơi có tổng bằng 0 là một biểu diễn toán học về tình huống trong đó mỗi người tham gia được hoặc mất tiện ích được cân bằng chính xác bởi những mất mát hoặc lợi ích của những người tham gia khác. Nếu tổng lợi ích của những người tham gia được cộng lại và tính tổng thiệt hại, cả hai sẽ có tổng bằng không. Cờ vua là một trò chơi có tổng bằng 0, nghĩa là cả hai bên đều có thể thắng, thua và hòa. Nếu người chơi A có thể thắng trong 1 nước đi thì nước đi đó chính là nước đi tốt nhất cho A. Nếu người B biết rằng có một nước đi mà dẫn đến tình huống người A có thể thắng ngay ở nước đi tiếp theo, trong khi nước đi khác thì sẽ dẫn đến tình huống mà người chơi A chỉ có thể tốt nhất là hòa, thì nước đi tốt nhất của người B chính là nước đi sau.

Theo lý thuyết này, bất kỳ lợi thế nào mà Người chơi A đạt được đều có nghĩa là bất lợi cho Người chơi B. Lợi thế có thể đến từ việc bắt quân của đối thủ hoặc có quân ở vị trí thuận lợi. Từ góc độ của Chess bot, khi bị mất quân đồng nghĩa với việc bị trừ điểm lợi thế, ăn được quân đồng nghĩa với cộng thêm điểm lợi thế. Từ ý tưởng này, ta sẽ xây dựng các thuật toán đánh giá lợi thế dựa trên mỗi nước đi cho Chess bot, khi tính toán đến việc đi một nước đi nào đó, Chess bot sẽ xem xét nước đi đó có lợi hay bất lợi trong tương lai (tùy theo độ sâu), từ đó đưa ra nước đi thích hợp.

- Các thuật toán để đưa ra nước đi tối ưu:
    + Lượng giá quân cờ, đánh giá thế trận hiện tại dựa trên vị trí các quân cờ: Đơn vị đo chính cho các thuật toán khác vì mục đích cuối cùng của Chess bot là đưa ra được nước đi có lợi thế nhiều nhất.
    + Thuật toán tìm kiếm đệ quy mở rộng không gian trạng thái: Với mỗi một nước đi của bản thân hoặc đối thủ, cần tính toán đến những khả năng tiếp theo trong tương lai mà đối thủ hoặc bản thân có thể đi tiếp (trạng thái) để đánh giá các trạng thái xem trạng thái nào ít tổn thất nhất. Chess bot cần phải tính toán đến các nước đi có thể của đối thủ cùng các khả năng có thể xảy ra trong tương lai thông qua không gian trạng thái, giúp cho nó có thể đưa ra nước đi tốt do đã tính toán được trước các lợi thế có thể có một cách tương đối sâu.
    + Thuật toán Minimax: Dựa trên những ý tưởng trên, cần có một thuật toán để có thể “tối đa hóa lợi thế” và “tối thiểu hóa bất lợi”. Giải thuật Minimax giúp tìm ra nước đi tốt nhất, bằng cách đi ngược từ cuối trò chơi trở về đầu. Tại mỗi bước, nó sẽ ước định rằng người A đang cố gắng tối đa hóa cơ hội thắng của A khi đến phiên anh ta, còn ở nước đi kế tiếp thì người chơi B cố gắng để tối thiểu hóa cơ hội thắng của người A (nghĩa là tối đa hóa cơ hội thắng của B). Hơn nữa, người A và người B phải tính toán được đến cả những nước đi có thể khiến bản thân rơi vào tình huống bất lợi nhất, hay gọi là tối thiểu hóa cơ hội thắng của chính mình, vì đó là nước đi mà đối thủ muốn đi nhất và phải tránh được điều đó xảy ra. Thuật toán Minimax sẽ đưa ra trạng thái hoặc là tốt nhất cho bản thân và tồi tệ nhất cho đối thủ.
    + Thuật toán cắt tỉa Alpha-Beta: Như đã nói, Chessbot cần phải sinh ra các không gian trạng thái bằng giải thuật đệ quy và đánh giá chúng bằng Minimax. Hiện tại, nó đã có thể đưa ra những quyết định đúng đắn và hợp lý. Tuy nhiên, độ sâu tìm kiếm trạng thái càng cao, thời gian tìm kiếm sẽ càng tăng lên (chỉ cần tìm kiếm với độ sâu là 4 thì thời gian có thể lên đến hàng trăm giây). Cắt tỉa alpha-beta giúp cải thiện hiệu quả của thuật toán bằng cách 'cắt tỉa' các nhánh mà Chessbot không cần đánh giá.

2.4. Thiết kế logic cho chess bot

Để Chessbot đưa ra một nước đi đủ tốt, nó cần một “con số” để hiểu được trạng thái bàn cờ hiện tại, chính xác hơn là một chức năng đánh giá. Về cơ bản, ta muốn gán một 'điểm số' cho từng trường hợp bàn cờ (tức là từng tập hợp vị trí của các quân cờ trên bàn cờ) để Chessbot có thể đưa ra quyết định về vị trí nào thuận lợi hơn các vị trí khác.

$$f\left(\right.$$  $$\left.\right) = 5$$

Khía cạnh đầu tiên và là nguyên tử cho quá trình đánh giá liên quan đến việc ấn định trọng số cho từng quân cờ. Nếu Chessbot chơi từ góc nhìn của quân đen, vậy thì bất kỳ quân đen nào sẽ cộng vào điểm số lợi thế của Chessbot, trong khi bất kỳ quân trắng nào sẽ trừ vào điểm lợi thế, theo các trọng số sau, tạm gọi là hệ số lực của mỗi quân cờ:

Tốt (Pawn): 100
Mã (Knight): 280
Tượng (Bishop): 320
Xe (Rook): 479
Hậu (Queen): 929
Vua (King): 60,000

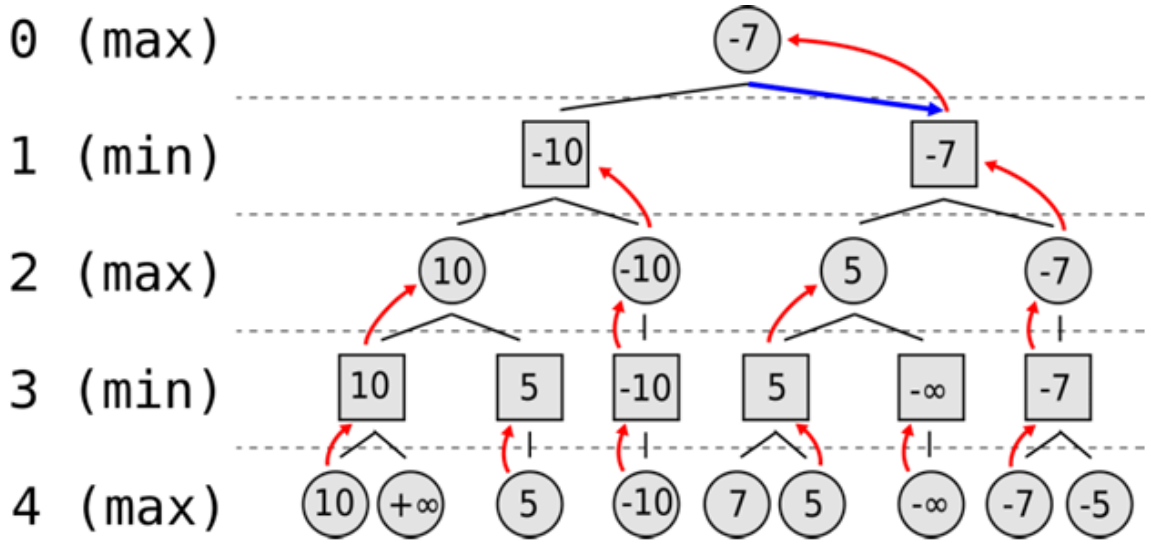
Các quân cờ đã được “đánh giá sức mạnh”, tuy nhiên, một quân cờ khi ở một trí cụ thể trên bàn cờ sẽ mạnh hơn ở các vị trí khác, ví dụ quân Mã trắng khi ở f3 có thể kiểm soát được 6 ô, và quan trọng nhất là các ô trung tâm, từ đó làm chậm các cuộc tấn công của quân đen vào khu trung tâm. Rõ ràng quân Mã trắng khi đứng ở f3 mạnh hơn hẳn so với việc nó đứng ở g1 (chỉ kiểm soát được 2 ô ở góc bàn cờ). Thậm chí khi đứng ở g1, số điểm lợi thế của người chơi quân trắng còn bị trừ đi vì lúc này quân Mã khá “vô dụng”. Vậy nên, cần đánh giá một quân cờ không chỉ ở giá trị lực của bản thân nó, mà phải đánh giá cả việc nó đang đứng ở đâu trên bàn cờ, vị trí đó tăng bao nhiêu lợi thế cho người chơi và Chessbot. Các giá trị sau đây thể hiện giá trị cộng thêm của một quân cờ khi đứng ở một vị trí cụ thể (đối với quân trắng), được gọi là bảng giá trị vị trí (Piece Square Tables), các giá trị này và giá trị các quân cờ đã trình bày ở trên được tham khảo dựa theo phần mềm chơi cờ vua nổi tiếng Sunfish Engine:

```
var pst_w = {
  p: [
    [100, 100, 100, 100, 105, 100, 100, 100],
    [78, 83, 86, 73, 102, 82, 85, 90],
    [7, 29, 21, 44, 40, 31, 44, 7],
    [-17, 16, -2, 15, 14, 0, 15, -13],
    [-26, 3, 10, 9, 6, 1, 0, -23],
    [-22, 9, 5, -11, -10, -2, 3, -19],
    [-31, 8, -7, -37, -36, -14, 3, -31],
    [0, 0, 0, 0, 0, 0, 0, 0],
  ],
  n: [
    [-66, -53, -75, -75, -10, -55, -58, -70],
    [-3, -6, 100, -36, 4, 62, -4, -14],
    [10, 67, 1, 74, 73, 27, 62, -2],
    [24, 24, 45, 37, 33, 41, 25, 17],
    [-1, 5, 31, 21, 22, 35, 2, 0],
    [-18, 10, 13, 22, 18, 15, 11, -14],
    [-23, -15, 2, 0, 2, 0, -23, -20],
    [-74, -23, -26, -24, -19, -35, -22, -69],
  ],
  b: [
    [-59, -78, -82, -76, -23, -107, -37, -50],
    [-11, 20, 35, -42, -39, 31, 2, -22],
    [-9, 39, -32, 41, 52, -10, 28, -14],
    [25, 17, 20, 34, 26, 25, 15, 10],
    [13, 10, 17, 23, 17, 16, 0, 7],
    [14, 25, 24, 15, 8, 25, 20, 15],
    [19, 20, 11, 6, 7, 6, 20, 16],
    [-7, 2, -15, -12, -14, -15, -10, -10],
  ],
  r: [
    [35, 29, 33, 4, 37, 33, 56, 50],
    [55, 29, 56, 67, 55, 62, 34, 60],
    [19, 35, 28, 33, 45, 27, 25, 15],
    [0, 5, 16, 13, 18, -4, -9, -6],
    [-28, -35, -16, -21, -13, -29, -46, -30],
    [-42, -28, -42, -25, -25, -35, -26, -46],
    [-53, -38, -31, -26, -29, -43, -44, -53],
    [-30, -24, -18, 5, -2, -18, -31, -32],
  ],
  q: [
    [6, 1, -8, -104, 69, 24, 88, 26],
    [14, 32, 60, -10, 20, 76, 57, 24],
    [-2, 43, 32, 60, 72, 63, 43, 2],
    [1, -16, 22, 17, 25, 20, -13, -6],
    [-14, -15, -2, -5, -1, -10, -20, -22],
    [-30, -6, -13, -11, -16, -11, -16, -27],
    [-36, -18, 0, -19, -15, -15, -21, -38],
    [-39, -30, -31, -13, -31, -36, -34, -42],
  ],
```

```
  k: [
    [4, 54, 47, -99, -99, 60, 83, -62],
    [-32, 10, 55, 56, 56, 55, 10, 3],
    [-62, 12, -57, 44, -67, 28, 37, -31],
    [-55, 50, 11, -4, -19, 13, 0, -49],
    [-55, -43, -52, -28, -51, -47, -8, -50],
    [-47, -42, -43, -79, -64, -32, -29, -32],
    [-4, 3, -14, -50, -57, -18, 13, 4],
    [17, 30, -3, -14, 6, -1, 40, 18],
  ],

  // Endgame King Table
  k_e: [
    [-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],
  ],
};
var pst_b = {
  p: pst_w['p'].slice().reverse(),
  n: pst_w['n'].slice().reverse(),
  b: pst_w['b'].slice().reverse(),
  r: pst_w['r'].slice().reverse(),
  q: pst_w['q'].slice().reverse(),
  k: pst_w['k'].slice().reverse(),
  k_e: pst_w['k_e'].slice().reverse(),
};
```

2.5. Thuật toán Minimax

Về cơ bản, minimax nhằm mục đích giảm thiểu những tổn thất có thể xảy ra, giả định rằng cả hai người chơi đều là những người đưa ra quyết định hợp lý. Chúng ta có thể biểu diễn các bước di chuyển có thể dưới dạng cây trò chơi, trong đó nốt bên trái của mỗi nốt cha là điểm số tối thiểu của “người chơi tối đa hóa”, còn nốt bên phải là điểm số tối đa của người chơi tối thiểu hóa. A là người chơi tối đa hóa, cố gắng tối đa hóa điểm số của mình, trong khi B là người chơi tối thiểu hóa, cố gắng giảm thiểu điểm số của A. Tại các nút lá, điểm đánh giá lợi thế được truy ngược lại gốc (Minimax giúp tìm ra nước đi tốt nhất bằng cách đi ngược từ cuối trò chơi trở về đầu). Vô cực dương và âm tương ứng là thắng và thua. Nút ở tầng 0 là trạng thái trò chơi hiện tại và mục tiêu là tối đa hóa điểm số của A.



Thuật toán minimax cần các tham số sau:

```
function minimax(game, depth, alpha, beta, isMaximizingPlayer, sum, color)
```

- chess: đối tượng Chess, để thực hiện các phương thức của đối tượng này như sinh các nước đi có thể, thực hiện nước đi, undo,…).
- depth: độ sâu tối đa của cây trò chơi.
- isMaximizingPlayer: biến boolean cho biết đến lượt của người chơi tối đa hay tối thiểu.
- sum: giá trị đánh giá lợi thế của thế cờ hiện tại.
- color: màu quân của người chơi hiện tại.

Thuật toán minimax đánh giá và tìm kiếm nước đi tốt nhất từ nước đi hiện tại, nghĩa là mỗi một lần gọi hàm là một nước đi. Vậy nên mỗi khi gọi hàm, ta sẽ tăng biến đếm số nước đi đã tính toán lên 1.

```
positionCount++;
```

Khai báo biến currentMove để lưu trữ nước đi đang xét hiện tại.

```
var currMove;
```

Từ đối tượng chess, sinh các nước đi hợp lệ tính từ vị trí hiện tại trên cây trò chơi cùng với các thông số chi tiết về nước đi đó và lưu vào một biến children.

```
var children = game.ugly_moves({ verbose: true });
```

Lưu ý, phải xáo trộn ngẫu nhiên mảng các nước đi trong children, để tránh việc luôn luôn chọn cùng một nước đi trong trường hợp có nhiều nước đi có giá trị minimax bằng nhau vì trong thuật toán minimax, nếu các giá trị đánh giá bằng nhau thì nó sẽ chọn một nước đi ngẫu nhiên từ các nước có giá trị đánh giá bằng nhau đó. Để tránh tình trạng lặp lại các lựa chọn trong trường hợp các giá trị đánh giá bằng nhau, ta sử dụng hàm sort để sắp xếp ngẫu nhiên các lựa chọn đó. Hàm sort này sắp xếp các phần tử trong một mảng, và được cài đặt ở đây bằng cách so sánh các phần tử ngẫu nhiên với nhau, trả về một giá trị ngẫu nhiên nằm trong khoảng [-0.5, 0.5], đảm bảo thứ tự của các phần tử bị thay đổi một cách ngẫu nhiên.

```
children.sort(function (a, b) {
  return 0.5 - Math.random();
});
```

Tiếp theo, kiểm tra nếu độ sâu tìm kiếm minimax đã đạt tới giới hạn (depth = 0) hoặc trạng thái hiện tại của trò chơi không có nước đi nào khả dĩ, trả về một mảng với giá trị null và giá trị sum hiện tại.

```
if (depth === 0 || children.length === 0) {
  return [null, sum];
}
```

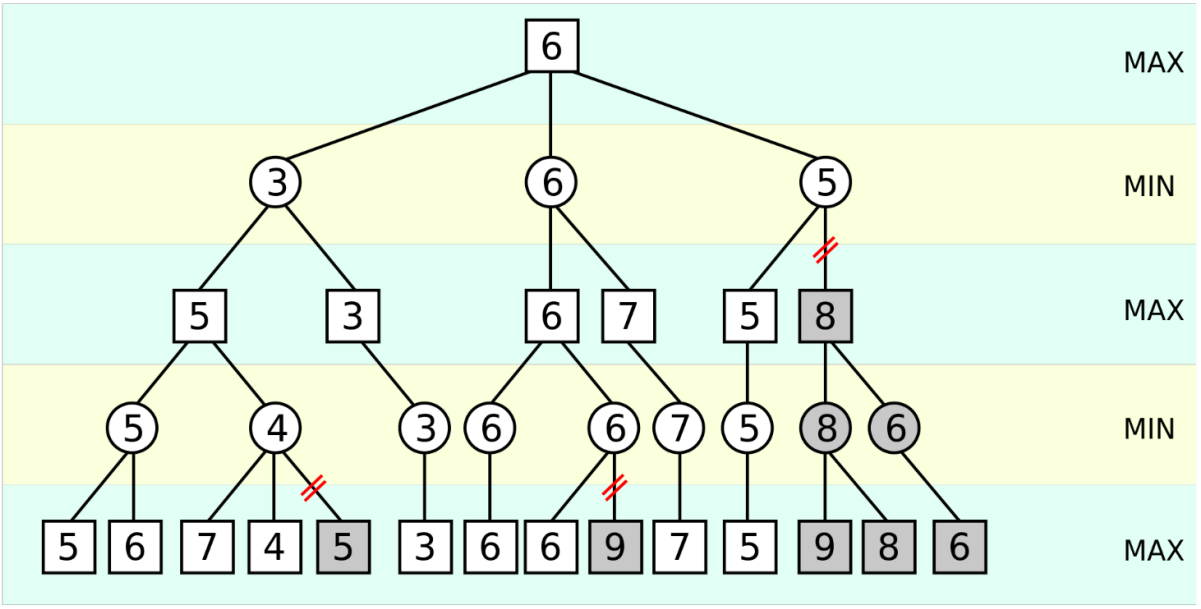
Tiếp đến là công việc chính của thuật toán minimax: Tìm ra nước đi "tốt nhất" hoặc "xấu nhất", và nó sẽ tìm trong danh sách các nước đi có thể từ vị trí hiện tại. Các biến lưu giá trị của nước đi tốt nhất và xấu nhất sẽ được khởi tạo bằng một giá trị được coi là tồi tệ nhất cho cả 2 người chơi: Biến maxValue ban đầu được gán bằng giá trị âm vô cùng, còn biến minValue được gán bằng giá trị dương vô cùng. Sau đó, một vòng lặp for được sử dụng để duyệt qua tất cả các nước đi con được sinh ra trong mảng children. Trong mỗi vòng lặp, currMove được gán bằng nước đi hiện tại, sau đó thuật toán lại tiếp tục tạo một bản sao của trò chơi bằng cách thực hiện nước đi này và lấy ra giá trị lợi thế của trạng thái mới bằng cách sử dụng hàm evaluateBoard(). Sau đó, thuật toán được gọi đệ quy với độ sâu giảm đi 1, đảo ngược giá trị isMaximizingPlayer(vì trò chơi đối kháng, 1 người là "tối đa hóa", người còn lại là "tối thiểu hóa"), và giá trị lợi thế sum mới. Kết quả là nước đi tốt nhất và giá trị lợi thế tại thế cờ mà nước đi tốt nhất đó được đi, chúng được lưu trữ trong các biến childBestMove và biến childValue.

```
var maxValue = Number.NEGATIVE_INFINITY;
var minValue = Number.POSITIVE_INFINITY;
var bestMove;
for (var i = 0; i < children.length; i++) {
  currMove = children[i];


  var currPrettyMove = game.ugly_move(currMove);
  var newSum = evaluateBoard(game, currPrettyMove, sum, color);
  var [childBestMove, childValue] = minimax(
    game,
    depth - 1,
    alpha,
    beta,
    !isMaximizingPlayer,
    newSum,
    color
  );

  game.undo();
```

```
    if (isMaximizingPlayer) {
      if (childValue > maxValue) {
        maxValue = childValue;
        bestMove = currPrettyMove;
      }
      if (childValue > alpha) {
        alpha = childValue;
      }
    } else {
      if (childValue < minValue) {
        minValue = childValue;
        bestMove = currPrettyMove;
      }
      if (childValue < beta) {
        beta = childValue;
      }
    }
```

Một cách ngắn gọn, hàm minimax trong game thực hiện công việc sau: Đầu tiên, các bước đi có thể được thực hiện từ trạng thái hiện tại của trò chơi được lưu trong biến "chess" bằng cách gọi phương thức "ugly_moves" và lưu vào biến "children". Sau đó, các bước đi trong "children" được sắp xếp ngẫu nhiên để tránh việc chọn cùng một bước đi khi có nhiều lựa chọn tốt. Tiếp theo, thuật toán sẽ tìm giá trị tốt nhất hoặc xấu nhất từ danh sách các bước đi trong "children". Đối với mỗi bước đi, nó sẽ lấy trạng thái của trò chơi sau khi thực hiện bước đi đó bằng cách gọi phương thức "ugly_move" và lưu vào biến "currPrettyMove". Sau đó, hàm "evaluateBoard" được gọi để đánh giá trạng thái mới của trò chơi và trả về một giá trị số được lưu trong biến "newSum". Hàm "minimax" được đệ quy gọi với các tham số mới, bao gồm trạng thái mới của trò chơi và giá trị số mới này. Tham số "depth" cũng được giảm xuống để thuật toán sẽ dừng lại sau một số bước cố định để tránh việc đệ quy quá sâu. Khi thuật toán đệ quy đến độ sâu tối đa hoặc trò chơi đã kết thúc, nó sẽ trả về một cặp giá trị, trong đó giá trị thứ hai được sử dụng để lựa chọn bước đi tốt nhất. Nếu "isMaximizingPlayer" đang đánh giá cho người chơi cần tối đa hóa giá trị, thuật toán sẽ chọn giá trị lớn nhất và lưu lại bước đi tương ứng trong biến "bestMove". Nếu "isMaximizingPlayer" đang đánh giá cho người chơi cần tối thiểu hóa giá trị, thuật toán sẽ chọn giá trị nhỏ nhất và lưu lại bước đi tương ứng trong biến "bestMove". Cuối cùng, thuật toán sẽ trả về cặp giá trị (bước đi tốt nhất và giá trị tương ứng) để sử dụng cho bước đi tiếp theo trong trò chơi.

2.6. Thuật toán cắt tỉa Alpha-Beta



Ý tưởng chính của thuật toán Alpha-Beta: Thuật toán duy trì hai giá trị, alpha và beta, tương ứng đại diện cho số điểm tối thiểu mà "người chơi tối đa hóa" (ở ví dụ trên là người chơi A, vì người chơi A đang cố gắng "tối đa hóa" cơ hội chiến thắng của anh ta) đang có và số điểm tối đa mà "người chơi tối thiểu hóa" (ở ví dụ trên là người chơi B, vì người chơi B đang cố gắng "tối thiểu hóa" cơ hội chiến thắng của A) đang có, hay ngắn gọn alpha là cơ hội chiến thắng của người A và beta là nguy cơ thua cuộc của người B. Ban đầu, alpha là âm vô cùng và beta là dương vô cùng, tức là cả hai người chơi đều bắt đầu với số điểm kém nhất có thể của họ. Bất cứ khi nào số điểm tối đa mà người chơi tối thiểu hóa (tức là người chơi "beta") đang có thấp hơn số điểm tối thiểu mà người chơi tối đa hóa (tức là người chơi "alpha") đang có (tức là beta < alpha), thì thuật toán sẽ bỏ qua, không cần đánh giá trạng thái này vì nếu từ trạng thái mà sinh ra tiếp các trạng thái tiếp theo thì nước đi sẽ luôn luôn tệ hơn.

Nói cách khác, các trạng thái trò chơi của cây tìm kiếm được phân tích để tìm kiếm nước đi tối ưu cho người chơi tối đa và tối thiểu. Vì vậy, nếu giải pháp được tìm thấy ở một nút con bên dưới là tốt hơn so với giải pháp tại nút cha, nút cha sẽ chọn giải pháp tốt nhất từ giải pháp của nút con đó. Điều này được thực hiện bằng cách theo dõi giá trị tốt nhất đã tìm thấy cho người chơi tối đa (maximizing player) hoặc người chơi tối thiểu (minimizing player) tại mỗi nút. Tuy nhiên, với Alpha-Beta pruning, một số nút không cần phân tích bởi vì chúng không ảnh hưởng đến giá trị tốt nhất đã tìm thấy của nút cha. Vì vậy, nếu ta đã tìm được một giá trị beta mà bé hơn hoặc bằng giá trị alpha, chúng ta sẽ không phải phân tích các nút con khác trong cây tìm kiếm, vì chúng ta sẽ không sử dụng kết quả của các nút con đó. Nếu đang xét đến người chơi tối đa, giá trị childValue được so sánh với giá trị hiện tại của maxValue. Nếu childValue lớn hơn maxValue, thì maxValue và bestMove được cập nhật để lưu trữ giá trị và nước đi tốt nhất cho người chơi tối đa tại nút hiện tại. Tương tự, giá trị childValue cũng được so sánh với alpha, và nếu childValue lớn hơn alpha, thì alpha được cập nhật để lưu trữ giá trị tốt nhất đã tìm thấy cho người chơi tối đa trên tất cả các nút con đã xét. Nếu đang xét đến người chơi tối thiểu, giá trị childValue được so sánh với giá trị hiện tại của minValue. Nếu childValue nhỏ hơn minValue, thì minValue và bestMove được cập nhật để lưu trữ giá trị và nước đi tốt nhất cho người chơi tối thiểu tại nút hiện tại. Khi một

node trên cây trò chơi được đánh giá, các giá trị alpha và beta được cập nhật tương ứng với vai trò của người chơi. Khi alpha của người chơi tối đa lớn hơn hoặc bằng beta của người chơi tối thiểu, cây con được ngắt và giá trị hiện tại của node được trả về. Nếu điều kiện alpha >= beta được thỏa mãn, nó có nghĩa là người chơi tối đa sẽ không chọn bất kỳ node con nào của node hiện tại, vì nếu chọn bất kỳ node con nào thì người chơi tối thiểu sẽ có thể chọn giá trị lớn hơn, do đó giá trị hiện tại của node sẽ không tốt hơn. Tương tự với người chơi tối thiểu. Việc cắt tỉa như vậy sẽ giảm thiểu số lượng nút mà thuật toán cần phân tích, giúp tăng tốc độ tính toán và giảm bớt tài nguyên sử dụng.

```
if (alpha >= beta) {
      break;
    }
  }

  if (isMaximizingPlayer) {
    return [bestMove, maxValue];
  } else {
    return [bestMove, minValue];
  }
}
```

2.7. Chạy chương trình và demo

Giao diện khi chạy chương trình

Tài liệu tham khảo

Giáo trình Artificial Intelligence – Stuart Russell, Peter Norvig 1994

Slide Trí tuệ nhân tạo - Gv Trần Thế Hùng

https://askeplaat.wordpress.com/534-2/mtdf-algorithm/

https://www.chessprogramming.org/Evaluation

https://www.chessprogramming.org/0x88

https://github.com/thomasahle/sunfish

https://www.chessprogramming.org/Main_Page