**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**TÊN HỌC PHẦN: Trí tuệ nhân tạo**

**ĐỀ TÀI: Thuật toán cờ vua AI**

**Giáo viên hướng dẫn:**

**Sinh viên thực hiện:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Stt** | **Mã sv** | **Họ và tên** | **Lớp** |
| 1 | 1771020611 | Nguyễn Viết Tâm | 17-11 |
| 2 | 1771020726 | Hoàng Khắc Tùng | 17-11 |

**Hà Nội, năm 2025**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**TÊN HỌC PHẦN: Trí tuệ nhân tạo**

**ĐỀ TÀI: Thuật toán cờ vua AI**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Mã Sinh Viên** | **Họ và Tên** | **Ngày Sinh** | **Điểm** | |
| **Bằng Số** | **Bằng Chữ** |
| 1 | 1771020611 | Nguyễn Viết Tâm | 09.11.2005 |  |  |
| 2 | 1771020726 | Hoàng Khắc Tùng |  |  |  |

CÁN BỘ CHẤM THI

**Hà Nội, năm 2025**

**LỜI CẢM ƠN**

*Trước hết, chúng em xin gửi lời tri ân chân thành tới giảng viên môn trí tuệ nhân tạo. Thầy không chỉ là người hướng dẫn mà còn là nguồn động viên và sự hỗ trợ quan trọng trong suốt quá trình học tập và nghiên cứu của chúng em. Sự quan tâm tận tâm và lòng nhiệt huyết của thầy đã giúp chúng em không chỉ thu được kiến thức chuyên ngành mà còn những góc nhìn thực tế quý báu và lời khuyên quý giá.*

*Phương pháp giảng dạy của thầy không chỉ giúp chúng em tiếp cận với tri thức một cách hiệu quả mà còn tạo ra cơ hội cho chúng em để phát triển khả năng và kiến thức cá nhân. Những buổi thảo luận và hoạt động nhóm không chỉ là nơi chúng em học hỏi kiến thức mới mẻ mà còn là môi trường để rèn luyện sự tự tin, kỹ năng giao tiếp và làm việc nhóm - những kỹ năng mà chúng em tin rằng sẽ rất quan trọng khi chúng em bước vào vào môi trường làm việc sau này.*

*Dưới sự hướng dẫn của thầy và sự nỗ lực không ngừng của toàn bộ nhóm, chúng em đã kết hợp kiến thức thuần túy từ giảng đường với những kiến thức mới mẻ và kinh nghiệm cá nhân để hoàn thành báo cáo một cách xuất sắc. Chúng em trân trọng mọi góp ý, đánh giá chân thành từ thầy cô và các bạn để từ đó hoàn thiện đề tài của chúng em hơn nữa.*

*Chúng em xin chân thành cảm ơn!*

# MỤC LỤC

[MỤC LỤC 2](#_Toc191587600)

[Chương 1: Mở đầu 4](#_Toc191587601)

[1.1 Giới thiệu đề tài 4](#_Toc191587602)

[1.2 Mục tiêu của đề tài 4](#_Toc191587603)

[1.3 Cách tiếp cận và phương pháp nghiên cứu 5](#_Toc191587604)

[1.4 Kết quả dự kiến đạt được 5](#_Toc191587605)

[Chương 2: Tổng quan về trò chơi cờ vua và thuật toán 6](#_Toc191587606)

[2.1 Trò chơi cờ vua 6](#_Toc191587607)

[2.1.1 Giới thiệu 6](#_Toc191587608)

[2.1.2 Lịch sử 6](#_Toc191587609)

[2.1.3 Quy tắc cơ bản 7](#_Toc191587610)

[2.2 Cơ sở lý thuyết để xây dựng Chess bot 10](#_Toc191587611)

[2.2.1 Ý tưởng đưa ra nước đi tối ưu 10](#_Toc191587612)

[2.2.2 Các thuật toán để đưa ra nước đi tối ưu 10](#_Toc191587613)

[Chương 3: Cài đặt trò chơi Cờ vua và triển khai thuật toán Negamax với cắt tỉa Alpha-Beta 13](#_Toc191587614)

[3.1 Quản lý giao diện đồ họa và điều khiển trò chơi 13](#_Toc191587615)

[3.2. Quản lý logic cốt lõi của trò chơi cờ vua 19](#_Toc191587616)

[3.3. Tạo nước đi cho máy tính 24](#_Toc191587617)

[Chương 4: Kiểm thử và đánh giá chung 28](#_Toc191587618)

[4.1 Mục tiêu kiểm thử 28](#_Toc191587619)

[4.2 Phương pháp kiểm thử 28](#_Toc191587620)

[4.2.1 Kiểm thử chức năng từng thành phần 28](#_Toc191587621)

[4.2.2 Kiểm thử đối kháng giữa người và máy 28](#_Toc191587622)

[4.2.3 Kiểm thử hiệu suất 28](#_Toc191587623)

[4.3 Kết quả kiểm thử 28](#_Toc191587624)

[4.4 Đánh giá chung 29](#_Toc191587625)

[4.4.1 Ưu điểm 29](#_Toc191587626)

[4.4.2 Hạn chế 29](#_Toc191587627)

[4.4.3 Hướng phát triển 29](#_Toc191587628)

[Tài liệu tham khảo 30](#_Toc191587629)

**Danh mục hình ảnh**

Hình 1 9

Hình 2 9

Hình 3 11

Hình 4 13

# Chương 1: Mở đầu

## 1.1 Giới thiệu đề tài

Trong thời đại công nghệ ngày nay, việc ứng dụng trí tuệ nhân tạo vào lĩnh vực giải trí và game đang trở nên ngày càng phổ biến. Trò chơi cờ vua, với độ phức tạp và chiến thuật đa dạng của nó, là một môi trường lý tưởng để khám phá và thử nghiệm sức mạnh của các thuật toán trí tuệ nhân tạo. Trong tinh thần này, việc xây dựng một trò chơi cờ vua thông minh sử dụng thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta không chỉ là một thách thức kỹ thuật mà còn là một cơ hội để khám phá những khía cạnh mới của trí tuệ nhân tạo trong môi trường trò chơi.

## 1.2 Mục tiêu của đề tài

* Phát triển thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta để máy tính có thể đánh giá tình hình hiện tại và dự đoán các nước đi tiếp theo một cách linh hoạt.
* Tối ưu hóa hàm đánh giá để máy tính có khả năng nhận diện các tình huống chiến thuật đặc biệt và phản ứng một cách linh hoạt dựa trên đánh giá này.
* Thực hiện thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta với khả năng điều chỉnh độ sâu của cây trò chơi để tối ưu cho các cấp độ khó khác nhau.
* Phát triển hàm đánh giá chiến thuật để máy tính có khả năng nhận diện và đánh giá chiến thuật như chiến thuật mở, chiến thuật phòng thủ, và chiến thuật tấn công. Tối ưu hóa thuật toán để máy tính có thể điều chỉnh chiến thuật của mình dựa trên tình hình cụ thể trong trò chơi.
* Thêm tính ngẫu nhiên vào quá trình đưa ra quyết định của máy tính để tạo ra sự đa dạng và tính khó đoán trong lối chơi. Đảm bảo rằng máy tính không luôn chọn những nước đi dựa hoàn toàn vào tính toán mà còn có yếu tố ngẫu nhiên, tương tự như lối chơi của con người.
* Thực hiện các bài kiểm tra chiến lược để đảm bảo rằng máy tính có khả năng đối phó với nhiều chiến thuật khác nhau của người chơi. Thu thập dữ liệu từ các trận đấu để đánh giá khả năng chiến thuật và tính hiệu quả của thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta.

## 1.3 Cách tiếp cận và phương pháp nghiên cứu

* Thu thập tài liệu chi tiết về chiến thuật cờ vua để phát triển hàm đánh giá chiến thuật cho thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta.
* Khảo sát người chơi để hiểu rõ mong muốn và yêu cầu của họ đối với một trò chơi cờ vua máy tính.
* Sử dụng thông tin thu được để tối ưu hóa giao diện người chơi và tích hợp các tính năng được người chơi đánh giá cao.
* Sử dụng các nền tảng trực tuyến để chia sẻ và nhận phản hồi từ cộng đồng lập trình viên và người chơi.

## 1.4 Kết quả dự kiến đạt được

* Xây dựng một trò chơi cờ vua máy tính có khả năng chơi thông minh sử dụng thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta. Máy tính có khả năng đánh giá tình hình hiện tại và đưa ra những quyết định chiến thuật linh hoạt.
* Triển khai thuật toán Negamax kết hợp với cắt tỉa Alpha-Beta với các cải tiến như cắt tỉa alpha-beta để tối ưu hóa hiệu suất tính toán.
* Đảm bảo rằng trò chơi chạy mượt mà và có thể xử lý cây trò chơi lớn mà không gây giảm trải nghiệm người chơi.

# Chương 2: Tổng quan về trò chơi cờ vua và thuật toán

## 2.1 Trò chơi cờ vua

### 2.1.1 Giới thiệu

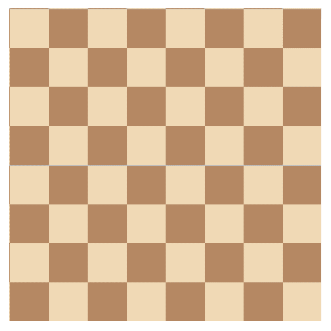
* Trò chơi cờ vua là một trò chơi chiến thuật hai người chơi trên một bảng cờ 8x8 ô, được chia thành 64 ô vuông đối lập. Mỗi người chơi điều khiển một đội quân, bao gồm vua, hậu, xe, tượng, mã, và tốt. Mục tiêu chính của trò chơi là đặt đối thủ vào tình thế "bắt bí" (checkmate), nghĩa là vua của đối thủ không thể di chuyển mà không bị chiếu.
* Mỗi quân cờ có cách di chuyển đặc trưng và quy tắc riêng, tạo nên sự đa dạng và phức tạp trong chiến thuật. Trò chơi cờ vua yêu cầu người chơi kết hợp sự tư duy chiến thuật và kỹ năng dự đoán đối thủ để đạt được chiến thắng. Đồng thời, nó còn mang lại cảm giác căng thẳng và hứng thú trong suốt quá trình di chuyển và chiến đấu trên bảng cờ.

### 2.1.2 Lịch sử

* **Xuất Hiện Ban Đầu:** Trò chơi cờ vua xuất hiện đầu tiên ở Ấn Độ khoảng thế kỷ thứ 6, với tên gọi là "chaturanga." Ban đầu, trò chơi này có những quân cờ đại diện cho các thành phần trong quân đội Ấn Độ cổ đại như tướng, mã, xe và tượng.
* **Phát Triển Ở Châu Âu:** Cờ vua được giới thiệu tại châu Âu thông qua các lộ trình thương mại và quân sự trong thời kỳ Trung Cổ. Qua các thế kỷ, quân cờ và luật chơi dần chuyển biến, tạo nên hình thức cờ vua hiện đại. Trò chơi trở thành một phần quan trọng của đời sống tinh thần và văn hóa ở châu Âu.
* **Thế Chiến Thế Giới và Cờ Vua**: Thế Chiến Thế Giới thường được biết đến với việc cờ vua trở thành một phần của cuộc sống hàng ngày và giải trí trong quân đội. Nhiều nhà vô địch cờ vua nổi tiếng đã xuất hiện trong thời kỳ này, như Alexander Alekhine và Bobby Fischer.
* **Cờ Vua Hiện Đại và Công Nghệ:** Với sự phát triển của công nghệ, cờ vua đã mở ra một thế giới mới thông qua trò chơi trực tuyến và máy tính chơi cờ. Các chương trình máy tính chơi cờ ngày càng thông minh, và thuật toán như Minimax đã được tích hợp để tăng cường khả năng chơi của máy tính.
* **Cờ Vua Như Một Môn Thể Thao:** Cờ vua không chỉ là một trò chơi trí tuệ mà còn là một môn thể thao với các giải đấu quốc tế như Chess Olympiad và World Chess Championship. Các đại kiện tướng cờ trở thành những người nổi tiếng, và cờ vua trở thành một sân chơi quốc tế đa dạng và thú vị.

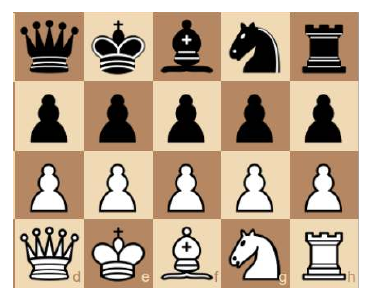
### 2.1.3 Quy tắc cơ bản

* Bàn Cờ: Bàn cờ cờ vua là ô vuông 8x8 với các ô được đánh số và đặt tên từ a đến h theo chiều ngang, và từ 1 đến 8 theo chiều dọc. Mỗi ô trên bàn cờ có thể là một vị trí cho một quân cờ.



Hình 1

* Quân Cờ và Vị Trí Ban Đầu: Mỗi bên (đen và trắng) có 16 quân cờ, bao gồm tướng, hậu, xe, tượng, mã, và tốt. Quân đen được đặt ở hàng số 7 và 8, trong khi quân trắng ở hàng số 1 và 2.



Hình 2

* Di Chuyển Cơ Bản:
* Vua (King): Di chuyển một ô theo chiều ngang, dọc, hoặc chéo. Vua không thể di chuyển vào ô đang bị đe dọa, và nếu bị bắt, trò chơi kết thúc.
* Hậu (Queen): Di chuyển bất kỳ số ô nào theo chiều ngang, dọc, hoặc chéo. Hậu kết hợp khả năng di chuyển của xe và tượng.
* Xe (Rook): Di chuyển bất kỳ số ô nào theo chiều ngang hoặc dọc. Không thể di chuyển chéo.
* Tượng (Bishop): Di chuyển bất kỳ số ô nào theo đường chéo. Không thể di chuyển theo chiều ngang hoặc dọc.
* Mã (Knight): Di chuyển theo hình chữ "L" với 2 bước theo một chiều và 1 bước theo chiều vuông góc. Có thể nhảy qua các quân cờ.
* Tốt (Pawn): Di chuyển về phía trước một ô, nhưng ở bước đầu tiên có thể di chuyển về phía trước hai ô. Ăn quân đối phương theo đường chéo. Khi tốt đến hàng số 8, nó có thể "phong tốt" thành quân cờ khác (trừ tướng và hậu).
* Bắt Quân Cờ: Khi một quân cờ di chuyển vào ô có quân cờ đối phương, quân đối phương bị bắt. Vua không thể bị bắt, nhưng nếu nó bị đe dọa và không có cách nào để di chuyển, đó là "hết cờ".
* Chiếu và hết cờ: Khi tướng bị đe dọa và không còn bước di chuyển hợp lệ nào khác, đó là "chiếu". Khi một bên không thể thực hiện bất kỳ bước di chuyển nào hợp lệ, đó là "hết cờ," và bên kia chiến thắng.



Hình 3

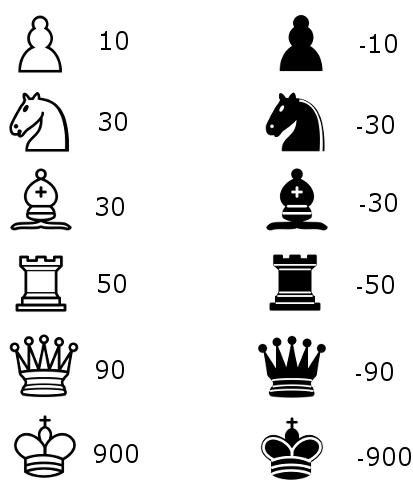
## 2.2 Cơ sở lý thuyết để xây dựng Chess bot

### 2.2.1 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.

### 2.2.2 Các thuật toán để đưa ra nước đi tối ưu

* Dựa vào ý tưởng trên, các thuật toán cần có để đưa ra được nước đi tối ưu cho Chess bot bao gồm:
* 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.



Hình 4

* 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.

# Chương 3: Cài đặt trò chơi Cờ vua và triển khai thuật toán Negamax với cắt tỉa Alpha-Beta

## 3.1 Quản lý giao diện đồ họa và điều khiển trò chơi

#Dùng pygame để tạo giao diện đồ họa

import pygame as p

import ChessEngine, ChessAI #Engine : quản lý logic chess,AI: tạo nước đi cho AI

import sys

from multiprocessing import Process, Queue

BOARD\_WIDTH = BOARD\_HEIGHT = 512

MOVE\_LOG\_PANEL\_WIDTH = 250 # độ dài của bảng ghi chép nước đi

MOVE\_LOG\_PANEL\_HEIGHT = BOARD\_HEIGHT

DIMENSION = 8 # số hàng,cột trên bàn cờ

SQUARE\_SIZE = BOARD\_HEIGHT // DIMENSION

MAX\_FPS = 15

IMAGES = {}

def loadImages():

# Hàm dùng để tải hình ảnh các quân cờ từ thư mục image có sẵn

    pieces = ['wp', 'wR', 'wN', 'wB', 'wK', 'wQ', 'bp', 'bR', 'bN', 'bB', 'bK', 'bQ']

    for piece in pieces:

        IMAGES[piece] = p.transform.scale(p.image.load("images/" + piece + ".png"), (SQUARE\_SIZE, SQUARE\_SIZE))

# transform.scale : điều chỉnh kích thước hình ảnh theo hình vuông

def main():

    p.init()

    screen = p.display.set\_mode((BOARD\_WIDTH + MOVE\_LOG\_PANEL\_WIDTH, BOARD\_HEIGHT)) # tạo cửa sổ trò chơi

    clock = p.time.Clock() # > kiểm soát tốc độ khung hình và thời gian

    screen.fill(p.Color("white")) #để board full trắngtrắng

    game\_state = ChessEngine.GameState()

    valid\_moves = game\_state.getValidMoves() # lấy list nước đi hợp lệ cho lượt hiên tại

    move\_made = False  # flag variable xem nước đi mới đã thực hiện chưa

    animate = False  # flag variable thực hiện hiệu ứng di chuyển khi nào

    loadImages()  #

    running = True

    square\_selected = ()

    player\_clicks = []

    game\_over = False

    ai\_thinking = False

    move\_undone = False

    move\_finder\_process = None

    move\_log\_font = p.font.SysFont("Arial", 14, False, False)

    player\_one = True

    player\_two = False

    while running:

        human\_turn = (game\_state.white\_to\_move and player\_one) or (not game\_state.white\_to\_move and player\_two)

        for e in p.event.get():

            if e.type == p.QUIT:

                p.quit()

                sys.exit()

            elif e.type == p.MOUSEBUTTONDOWN:

                if not game\_over:

                    location = p.mouse.get\_pos()

                    col = location[0] // SQUARE\_SIZE

                    row = location[1] // SQUARE\_SIZE

                    if square\_selected == (row, col) or col >= 8:

                        square\_selected = ()

                        player\_clicks = []

                    else:

                        square\_selected = (row, col)

                        player\_clicks.append(square\_selected)

                    if len(player\_clicks) == 2 and human\_turn:

                        move = ChessEngine.Move(player\_clicks[0], player\_clicks[1], game\_state.board)

                        for i in range(len(valid\_moves)):

                            if move == valid\_moves[i]:

                                game\_state.makeMove(valid\_moves[i])

                                move\_made = True

                                animate = True

                                square\_selected = ()

                                player\_clicks = []

                        if not move\_made:

                            player\_clicks = [square\_selected]

            # key handler

            elif e.type == p.KEYDOWN:

                if e.key == p.K\_z:

                    game\_state.undoMove()

                    move\_made = True

                    animate = False

                    game\_over = False

                    if ai\_thinking:

                        move\_finder\_process.terminate()

                        ai\_thinking = False

                    move\_undone = True

                if e.key == p.K\_r:

                    game\_state = ChessEngine.GameState()

                    valid\_moves = game\_state.getValidMoves()

                    square\_selected = ()

                    player\_clicks = []

                    move\_made = False

                    animate = False

                    game\_over = False

                    if ai\_thinking:

                        move\_finder\_process.terminate()

                        ai\_thinking = False

                    move\_undone = True

        if not game\_over and not human\_turn and not move\_undone:

            if not ai\_thinking:

                ai\_thinking = True

                return\_queue = Queue()

                move\_finder\_process = Process(target=ChessAI.findBestMove, args=(game\_state, valid\_moves, return\_queue))

                move\_finder\_process.start()

            if not move\_finder\_process.is\_alive():

                ai\_move = return\_queue.get()

                if ai\_move is None:

                    ai\_move = ChessAI.findRandomMove(valid\_moves)

                game\_state.makeMove(ai\_move)

                move\_made = True

                animate = True

                ai\_thinking = False

        if move\_made:

            if animate:

                animateMove(game\_state.move\_log[-1], screen, game\_state.board, clock)

            valid\_moves = game\_state.getValidMoves()

            move\_made = False

            animate = False

            move\_undone = False

        drawGameState(screen, game\_state, valid\_moves, square\_selected)

        if not game\_over:

            drawMoveLog(screen, game\_state, move\_log\_font)

        if game\_state.checkmate:

            game\_over = True

            if game\_state.white\_to\_move:

                drawEndGameText(screen, "Chiếu hết, đen thắng")

            else:

                drawEndGameText(screen, "Chiếu hết, trắng thắng")

        elif game\_state.stalemate:

            game\_over = True

            drawEndGameText(screen, "Stalemate")

        clock.tick(MAX\_FPS)

        p.display.flip()

def drawGameState(screen, game\_state, valid\_moves, square\_selected):

    drawBoard(screen)

    highlightSquares(screen, game\_state, valid\_moves, square\_selected)

    drawPieces(screen, game\_state.board)

def drawBoard(screen):

    global colors

    colors = [p.Color("white"), p.Color("gray")]

    for row in range(DIMENSION):

        for column in range(DIMENSION):

            color = colors[((row + column) % 2)]

            p.draw.rect(screen, color, p.Rect(column \* SQUARE\_SIZE, row \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE))

def highlightSquares(screen, game\_state, valid\_moves, square\_selected):

    if (len(game\_state.move\_log)) > 0:

        last\_move = game\_state.move\_log[-1]

        s = p.Surface((SQUARE\_SIZE, SQUARE\_SIZE))

        s.set\_alpha(100)

        s.fill(p.Color('green'))

        screen.blit(s, (last\_move.end\_col \* SQUARE\_SIZE, last\_move.end\_row \* SQUARE\_SIZE))

    if square\_selected != ():

        row, col = square\_selected

        if game\_state.board[row][col][0] == (

                'w' if game\_state.white\_to\_move else 'b'):

            s = p.Surface((SQUARE\_SIZE, SQUARE\_SIZE))

            s.set\_alpha(100)

            s.fill(p.Color('blue'))

            screen.blit(s, (col \* SQUARE\_SIZE, row \* SQUARE\_SIZE))

            s.fill(p.Color('yellow'))

            for move in valid\_moves:

                if move.start\_row == row and move.start\_col == col:

                    screen.blit(s, (move.end\_col \* SQUARE\_SIZE, move.end\_row \* SQUARE\_SIZE))

def drawPieces(screen, board):

    for row in range(DIMENSION):

        for column in range(DIMENSION):

            piece = board[row][column]

            if piece != "--":

                screen.blit(IMAGES[piece], p.Rect(column \* SQUARE\_SIZE, row \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE))

def drawMoveLog(screen, game\_state, font):

    move\_log\_rect = p.Rect(BOARD\_WIDTH, 0, MOVE\_LOG\_PANEL\_WIDTH, MOVE\_LOG\_PANEL\_HEIGHT)

    p.draw.rect(screen, p.Color('black'), move\_log\_rect)

    move\_log = game\_state.move\_log

    move\_texts = []

    for i in range(0, len(move\_log), 2):

        move\_string = str(i // 2 + 1) + '. ' + str(move\_log[i]) + " "

        if i + 1 < len(move\_log):

            move\_string += str(move\_log[i + 1]) + "  "

        move\_texts.append(move\_string)

    moves\_per\_row = 3

    padding = 5

    line\_spacing = 2

    text\_y = padding

    for i in range(0, len(move\_texts), moves\_per\_row):

        text = ""

        for j in range(moves\_per\_row):

            if i + j < len(move\_texts):

                text += move\_texts[i + j]

        text\_object = font.render(text, True, p.Color('white'))

        text\_location = move\_log\_rect.move(padding, text\_y)

        screen.blit(text\_object, text\_location)

        text\_y += text\_object.get\_height() + line\_spacing

def drawEndGameText(screen, text):

    font = p.font.SysFont("Helvetica", 32, True, False)

    text\_object = font.render(text, False, p.Color("gray"))

    text\_location = p.Rect(0, 0, BOARD\_WIDTH, BOARD\_HEIGHT).move(BOARD\_WIDTH / 2 - text\_object.get\_width() / 2,

                                                                 BOARD\_HEIGHT / 2 - text\_object.get\_height() / 2)

    screen.blit(text\_object, text\_location)

    text\_object = font.render(text, False, p.Color('black'))

    screen.blit(text\_object, text\_location.move(2, 2))

def animateMove(move, screen, board, clock):

    global colors

    d\_row = move.end\_row - move.start\_row

    d\_col = move.end\_col - move.start\_col

    frames\_per\_square = 10

    frame\_count = (abs(d\_row) + abs(d\_col)) \* frames\_per\_square

    for frame in range(frame\_count + 1):

        row, col = (move.start\_row + d\_row \* frame / frame\_count, move.start\_col + d\_col \* frame / frame\_count)

        drawBoard(screen)

        drawPieces(screen, board)

        color = colors[(move.end\_row + move.end\_col) % 2]

        end\_square = p.Rect(move.end\_col \* SQUARE\_SIZE, move.end\_row \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE)

        p.draw.rect(screen, color, end\_square)

        if move.piece\_captured != '--':

            if move.is\_enpassant\_move:

                enpassant\_row = move.end\_row + 1 if move.piece\_captured[0] == 'b' else move.end\_row - 1

                end\_square = p.Rect(move.end\_col \* SQUARE\_SIZE, enpassant\_row \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE)

            screen.blit(IMAGES[move.piece\_captured], end\_square)

        screen.blit(IMAGES[move.piece\_moved], p.Rect(col \* SQUARE\_SIZE, row \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE))

        p.display.flip()

        clock.tick(60)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Hiển thị bàn cờ** bằng pygame:

* Kích thước bàn cờ (BOARD\_WIDTH, BOARD\_HEIGHT).
* Tải ảnh quân cờ (loadImages()).
* Vẽ bàn cờ và các quân cờ (drawBoard(), drawPieces()).

**Xử lý tương tác của người chơi**:

* Nhận click chuột để chọn quân cờ và đi nước (MOUSEBUTTONDOWN).
* Nhận phím Z để hoàn tác (undoMove()).
* Nhận phím R để đặt lại ván cờ (reset game).

**Tích hợp AI** (từ ChessAI.py):

* Nếu lượt của AI, nó sẽ gọi findBestMove() để tìm nước đi tối ưu.
* AI chạy song song bằng multiprocessing để tránh làm chậm giao diện.

**Xử lý kết thúc ván cờ**:

* Hiển thị "Chiếu hết, Trắng thắng" hoặc "Chiếu hết, Đen thắng" nếu có checkmate.
* Hiển thị "Stalemate" nếu ván cờ hòa.

## 3.2. Quản lý logic cốt lõi của trò chơi cờ vua

* **Quản lý trạng thái bàn cờ (GameState)**

**Bảng cờ (self.board)**

Bàn cờ được biểu diễn dưới dạng một danh sách 2D (8x8), trong đó:

* "wp" = White Pawn (Tốt trắng)
* "bK" = Black King (Vua đen)
* "--" = Ô trống

Ví dụ:

self.board = [

    ["bR", "bN", "bB", "bQ", "bK", "bB", "bN", "bR"],

    ["bp", "bp", "bp", "bp", "bp", "bp", "bp", "bp"],

    ["--", "--", "--", "--", "--", "--", "--", "--"],

    ["--", "--", "--", "--", "--", "--", "--", "--"],

    ["--", "--", "--", "--", "--", "--", "--", "--"],

    ["--", "--", "--", "--", "--", "--", "--", "--"],

    ["wp", "wp", "wp", "wp", "wp", "wp", "wp", "wp"],

    ["wR", "wN", "wB", "wQ", "wK", "wB", "wN", "wR"]]

Mỗi ô cờ chứa **ký hiệu quân cờ** hoặc "--" nếu ô trống.

**Các trạng thái khác**

* self.white\_to\_move: True nếu đến lượt Trắng đi, False nếu đến lượt Đen.
* self.move\_log: Danh sách ghi lại tất cả các nước đi đã thực hiện.
* self.checkmate: True nếu ván cờ kết thúc vì chiếu hết.
* self.stalemate: True nếu ván cờ kết thúc vì bế tắc (không còn nước đi hợp lệ).
* **Xử lý nước đi**

**makeMove(move) – Thực hiện một nước đi**

Hàm này thay đổi trạng thái bàn cờ bằng cách:

1. Di chuyển quân cờ từ start\_row, start\_col đến end\_row, end\_col.
2. Cập nhật vị trí quân vua (nếu vua di chuyển).
3. Xử lý các trường hợp đặc biệt như: phong cấp tốt, bắt tốt qua đường, nhập thành.

Ví dụ:

def makeMove(self, move):

    self.board[move.start\_row][move.start\_col] = "--"

    self.board[move.end\_row][move.end\_col] = move.piece\_moved

    self.white\_to\_move = not self.white\_to\_move

    self.move\_log.append(move)

**undoMove() – Hoàn tác nước đi cuối cùng**

Hàm này sẽ lấy nước đi cuối cùng từ self.move\_log và hoàn tác nó, đưa bàn cờ về trạng thái trước đó.  
Ví dụ:

def undoMove(self):

    if len(self.move\_log) != 0:

        move = self.move\_log.pop()

        self.board[move.start\_row][move.start\_col] = move.piece\_moved

        self.board[move.end\_row][move.end\_col] = move.piece\_captured

        self.white\_to\_move = not self.white\_to\_move

Hàm này giúp người chơi quay lại nước đi trước nếu họ bấm **Undo**.

* **Xác định nước đi hợp lệ (getValidMoves())**

Hàm này xác định **tất cả nước đi hợp lệ** dựa trên luật cờ vua.

**Quá trình kiểm tra**

1. Lấy tất cả nước đi có thể thực hiện (getAllPossibleMoves()).
2. Loại bỏ những nước đi khiến vua bị chiếu.
3. Kiểm tra các điều kiện kết thúc ván cờ (chiếu hết, bế tắc).

Ví dụ:

def getValidMoves(self):

    moves = self.getAllPossibleMoves()

    for i in range(len(moves)-1, -1, -1):

        self.makeMove(moves[i])

        if self.inCheck():

            moves.remove(moves[i])

        self.undoMove()

    return moves

* **Kiểm tra chiếu (inCheck(), checkForPinsAndChecks())**

**inCheck() – Kiểm tra xem vua có bị chiếu không**

Hàm này kiểm tra xem vị trí của vua có bị quân đối thủ tấn công không.

def inCheck(self):

    if self.white\_to\_move:

        return self.squareUnderAttack(self.white\_king\_location[0], self.white\_king\_location[1])

    else:

        return self.squareUnderAttack(self.black\_king\_location[0], self.black\_king\_location[1])

**Nếu True, vua đang bị chiếu, cần tránh chiếu hết.**

**checkForPinsAndChecks() – Tìm các quân đang ghim hoặc chiếu vua**

* **Ghim (Pin)**: Một quân bị ghim khi di chuyển sẽ khiến vua bị chiếu.
* **Chiếu (Check)**: Một quân đang tấn công vua.
* **Tạo danh sách nước đi hợp lệ cho từng quân cờ**

Hàm này duyệt qua từng ô trên bàn cờ, kiểm tra loại quân cờ và gọi hàm tương ứng.

**Tốt (getPawnMoves())**

* Đi thẳng nếu ô trước trống.
* Ăn chéo quân đối thủ.
* Phong cấp khi đến hàng cuối.
* Bắt tốt qua đường (en passant).

def getPawnMoves(self, row, col, moves):

    if self.white\_to\_move:

        move\_amount = -1

        start\_row = 6

    else:

        move\_amount = 1

        start\_row = 1

    if self.board[row + move\_amount][col] == "--":

        moves.append(Move((row, col), (row + move\_amount, col), self.board))

**Xe (getRookMoves())**

* Đi ngang hoặc dọc nếu không bị cản.

def getRookMoves(self, row, col, moves):

    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for direction in directions:

        for i in range(1, 8):

            end\_row = row + direction[0] \* i

            end\_col = col + direction[1] \* i

            if 0 <= end\_row < 8 and 0 <= end\_col < 8:

                if self.board[end\_row][end\_col] == "--":

                    moves.append(Move((row, col), (end\_row, end\_col), self.board))

                else:

                    break

**Mã (getKnightMoves())**

* Đi theo hình chữ "L".
* Có thể nhảy qua quân khác.

def getKnightMoves(self, row, col, moves):

    knight\_moves = [(-2, -1), (-2, 1), (-1, 2), (1, 2), (2, -1), (2, 1), (-1, -2), (1, -2)]

    for move in knight\_moves:

        end\_row, end\_col = row + move[0], col + move[1]

        if 0 <= end\_row < 8 and 0 <= end\_col < 8:

            moves.append(Move((row, col), (end\_row, end\_col), self.board))

**Tượng (getBishopMoves())**

* Đi chéo không bị cản.

**Hậu (getQueenMoves())**

* Kết hợp **Xe** và **Tượng**.

**Vua (getKingMoves())**

* Đi 1 ô theo mọi hướng.
* Xử lý nhập thành (castling).

## 3.3. Tạo nước đi cho máy tính

**Dùng Negamax kết hợp với cắt tỉa Alpha-Beta** để tìm nước đi tốt nhất.

* + **Negamax** là một biến thể của Minimax, giúp tính điểm nước đi của AI và đối thủ bằng một công thức duy nhất:

score=−Negamax(baˋn cờ,độ saˆu−1,−β,−α,−hệ soˆˊ lượt đi)

**Cắt tỉa Alpha-Beta** giúp loại bỏ những nhánh tìm kiếm không cần thiết, giúp AI tính nhanh hơn.

* **Hàm chính: findBestMove()**

Hàm này gọi **Negamax + Alpha-Beta** để tìm nước đi tốt nhất.

def findBestMove(game\_state, valid\_moves, return\_queue):

    global next\_move

    next\_move = None

    random.shuffle(valid\_moves)  # Xáo trộn để tránh AI luôn đi theo một mẫu

    findMoveNegaMaxAlphaBeta(game\_state, valid\_moves, DEPTH, -CHECKMATE, CHECKMATE,

                             1 if game\_state.white\_to\_move else -1)

    return\_queue.put(next\_move)  # Trả về nước đi tốt nhất

**Giải thích:**

1. valid\_moves chứa danh sách các nước đi hợp lệ.
2. Gọi findMoveNegaMaxAlphaBeta() để tìm nước đi tốt nhất.
3. Lưu kết quả vào return\_queue để gửi về cho ChessMain.py.

* **Negamax với Alpha-Beta (findMoveNegaMaxAlphaBeta())**

Đây là thuật toán quan trọng nhất, giúp AI tìm ra nước đi mạnh nhất.

def findMoveNegaMaxAlphaBeta(game\_state, valid\_moves, depth, alpha, beta, turn\_multiplier):

    global next\_move

    if depth == 0:  # Nếu đạt đến độ sâu giới hạn, đánh giá bàn cờ

        return turn\_multiplier \* scoreBoard(game\_state)

    max\_score = -CHECKMATE  # Khởi tạo điểm số tệ nhất

    for move in valid\_moves:

        game\_state.makeMove(move)  # Thực hiện nước đi giả lập

        next\_moves = game\_state.getValidMoves()  # Lấy các nước đi tiếp theo

        score = -findMoveNegaMaxAlphaBeta(game\_state, next\_moves, depth - 1, -beta, -alpha, -turn\_multiplier)

        game\_state.undoMove()  # Hoàn tác nước đi sau khi tính điểm

        if score > max\_score:  # Nếu điểm số tốt hơn, cập nhật

            max\_score = score

            if depth == DEPTH:

                next\_move = move  # Lưu nước đi tốt nhất ở độ sâu gốc

        alpha = max(alpha, max\_score)  # Cập nhật Alpha

        if alpha >= beta:  # Cắt tỉa

            break

    return max\_score

**Giải thích:**

1. Nếu depth == 0, gọi scoreBoard() để đánh giá bàn cờ.
2. Duyệt tất cả nước đi hợp lệ (valid\_moves).
3. Giả lập từng nước đi (makeMove()), tính điểm bằng Negamax đệ quy, rồi hoàn tác (undoMove()).
4. Cập nhật nước đi tốt nhất (next\_move) nếu tìm được nước đi mạnh hơn.
5. Cắt tỉa Alpha-Beta để giảm số nước đi cần kiểm tra.

**Hàm đánh giá bàn cờ: scoreBoard()**

Hàm này giúp AI biết khi nào nó đang thắng hay thua.

def scoreBoard(game\_state):

    if game\_state.checkmate:

        return -CHECKMATE if game\_state.white\_to\_move else CHECKMATE

    elif game\_state.stalemate:

        return STALEMATE  # Hòa

    score = 0  # Khởi tạo điểm số

    for row in range(len(game\_state.board)):

        for col in range(len(game\_state.board[row])):

            piece = game\_state.board[row][col]

            if piece != "--":

                piece\_position\_score = piece\_position\_scores[piece][row][col] if piece[1] != "K" else 0

                if piece[0] == "w":

                    score += piece\_score[piece[1]] + piece\_position\_score

                else:

                    score -= piece\_score[piece[1]] + piece\_position\_score

    return score

**Giải thích:**

1. Nếu AI thắng (checkmate) → trả về +1000 điểm.
2. Nếu thua (checkmate của đối thủ) → trả về -1000 điểm.
3. Nếu hòa (stalemate) → trả về 0 điểm.
4. Nếu không, tính tổng giá trị quân cờ dựa trên piece\_score và vị trí trên bàn cờ (piece\_position\_scores).

* **Hàm findRandomMove()**

Một phiên bản đơn giản, chọn nước đi ngẫu nhiên (dùng khi AI không tìm thấy nước đi tốt).

def findRandomMove(valid\_moves):

    return random.choice(valid\_moves)

* **Điểm số quân cờ (piece\_score)**

Mỗi quân cờ có điểm số khác nhau, giúp AI đánh giá bàn cờ.

piece\_score = {"K": 0, "Q": 9, "R": 5, "B": 3, "N": 3, "p": 1}

**Quân hậu (Q)** = 9 điểm (mạnh nhất sau vua).  
**Xe (R)** = 5 điểm.  
**Tượng (B) & Mã (N)** = 3 điểm.  
**Tốt (p)** = 1 điểm.

* **Điểm số vị trí quân cờ (piece\_position\_scores)**

Không chỉ giá trị quân cờ quan trọng, mà **vị trí trên bàn cờ** cũng quyết định sức mạnh.

Ví dụ: **Mã (Knight) mạnh hơn khi ở trung tâm**:

knight\_scores = [

    [0.0, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.0],

    [0.1, 0.3, 0.5, 0.5, 0.5, 0.5, 0.3, 0.1],

    [0.2, 0.5, 0.6, 0.65, 0.65, 0.6, 0.5, 0.2],

    [0.2, 0.55, 0.65, 0.7, 0.7, 0.65, 0.55, 0.2],

    [0.2, 0.5, 0.65, 0.7, 0.7, 0.65, 0.5, 0.2],

    [0.2, 0.55, 0.6, 0.65, 0.65, 0.6, 0.55, 0.2],

    [0.1, 0.3, 0.5, 0.55, 0.55, 0.5, 0.3, 0.1],

    [0.0, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.0]]

Mã càng gần trung tâm, điểm càng cao, vì nó có thể kiểm soát nhiều ô hơn.

# Chương 4: Kiểm thử và đánh giá chung

## 4.1 Mục tiêu kiểm thử

Quá trình kiểm thử nhằm đảm bảo rằng hệ thống AI chơi cờ vua hoạt động chính xác theo các tiêu chí sau:  
- Tuân thủ luật chơi cờ vua (các nước đi hợp lệ, nhập thành, phong cấp, bắt tốt qua đường, v.v.).  
- Đánh giá đúng trạng thái bàn cờ và điểm số của từng nước đi.  
- AI có thể đưa ra nước đi hợp lý trong thời gian ngắn.  
- Xác minh thuật toán Negamax và cắt tỉa Alpha-Beta hoạt động hiệu quả.

## 4.2 Phương pháp kiểm thử

### 4.2.1 Kiểm thử chức năng từng thành phần

Các thành phần chính được kiểm thử bao gồm:  
- makeMove() & undoMove(): Kiểm tra xem AI có thể thực hiện và hoàn tác nước đi chính xác không.  
- getValidMoves(): Đảm bảo rằng AI chỉ chọn nước đi hợp lệ.  
- scoreBoard(): Xác minh điểm số bàn cờ phản ánh đúng lợi thế của mỗi bên.  
- findBestMove(): Đánh giá khả năng AI tìm ra nước đi tối ưu.

### 4.2.2 Kiểm thử đối kháng giữa người và máy

- Chơi thử với nhiều cấp độ người chơi để đánh giá khả năng của AI.  
- So sánh nước đi của AI với các engine cờ vua nổi tiếng như Stockfish.

### 4.2.3 Kiểm thử hiệu suất

- Đo thời gian AI tìm nước đi ở các độ sâu khác nhau.  
- Kiểm tra ảnh hưởng của cắt tỉa Alpha-Beta đến tốc độ tìm kiếm.

## 4.3 Kết quả kiểm thử

- Độ chính xác: AI thực hiện nước đi đúng luật, không có lỗi vi phạm luật cờ vua.  
- Hiệu suất: Khi sử dụng cắt tỉa Alpha-Beta, tốc độ tính toán được cải thiện đáng kể.  
- Khả năng đối kháng: AI có thể đánh bại người chơi ở mức trung bình nhưng gặp khó khăn trước người chơi mạnh.  
- Điểm yếu: AI chưa có khai cuộc và chiến thuật nâng cao, dễ mắc bẫy trong giai đoạn đầu ván cờ.

## 4.4 Đánh giá chung

### 4.4.1 Ưu điểm

- Hệ thống hoạt động ổn định, tuân thủ luật chơi.  
- Thuật toán Negamax với Alpha-Beta giúp AI tìm nước đi hợp lý trong thời gian ngắn.  
- Cách đánh giá bàn cờ bằng `scoreBoard()` giúp AI đưa ra quyết định hợp lý.

### 4.4.2 Hạn chế

- AI chưa có hệ thống khai cuộc mạnh, dễ bị thua sớm nếu gặp người chơi có chiến thuật tốt.  
- Độ sâu tìm kiếm còn hạn chế do hiệu suất tính toán.  
- AI chưa có khả năng học tập từ ván chơi trước (không có Machine Learning).

### 4.4.3 Hướng phát triển

- Cải thiện thuật toán đánh giá bàn cờ bằng cách thêm dữ liệu khai cuộc.  
- Áp dụng Machine Learning để giúp AI học từ ván đấu trước.  
- Nâng cấp thuật toán để giảm thời gian tính toán mà vẫn giữ chất lượng nước đi.

# 

# 

# Tài liệu tham khảo

* Nguyễn Thanh Tuấn, 08/2020, Sách Deep Learning cơ bản V2, [tr.109 – 147]:

(1) <https://nttuan8.com/sach-deep-learning-co-ban/>

(2) <https://nttuan8.com/bai-7-gioi-thieu-keras-va-bai-toan-phan-loai-anh/>

* TopDeV:

        https://topdev.vn/blog/huong-dan-tao-mot-doi-thu-ai-don-gian-cho-co-vua/

* Chessgame | Github.
* Minimax Algorithm | Wikipedia.
* Alpha – beta pruning | Wikipedia. 8. Algorithms Explained – minimax and alpha-beta pruning | Sebastian Lague