ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÀI TẬP LỚN
NHẬP MÔN TRÍ TUỆ NHÂN TẠO

GAME: CỜ VUA

NHÓM 06

Sinh viên thực hiện:

Phạm Việt Hoàng	20224854
Đoàn Mạnh Hùng	20224995
Phạm Thanh An	20224911
Nguyễn Thị Trà My	20225049
Đồng Văn Hiếu	20224980

Giảng viên hướng dẫn: TS. Đỗ Tiến Dũng

Hà Nội, Ngày 18 tháng 12 năm 2024

MỤC LỤC

DANH MỤC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU VÀ MÔ TẢ BÀI TOÁN

1.1 Bài toán thực tế cần giải quyết

Ngày nay, cùng với sự phát triển dữ dội của Trí tuệ nhân tạo, ngành công nghệ Thông tin đang tạo ra nhiều đổi mới mang nhiều giá trị cho xã hội. Trí tuệ nhân tạo có đặc tính nổi bật là khả năng phân tích và đưa ra quyết định(make decision), từ đó có thể tư duy độc lập với con người.

Cờ vua là một trò chơi bàn cổ điển phổ biến, được chơi trên một bảng ô vuông 8x8 với 64 ô. Trò chơi này được chia thành 2 phe: trắng và đen, mang tính đối kháng cạnh tranh cao, mang nhiều giá trị về mặt tư duy. Hiện tại, đây là môn cờ phổ biến nhất thế giới với cộng đồng người chơi mạnh.

Xuất phát từ điều trên, cùng với việc đã có một số tổ chức ứng dụng trí tuệ nhân tạo trong cờ vua giúp đề tài có thể dễ dàng phổ biến và kiểm thử, nhóm 03 trong học phần Nhập môn Trí tuệ Nhân tạo đã quyết định chọn đề tài: Xây dựng trò chơi cờ vua ứng dụng trí tuệ nhân tạo

1.2 Mục tiêu, kết quả mong muốn

1.2.1 Mục tiêu

Mục tiêu trong dự án này của nhóm là xây dựng website chơi cờ vua với ba chức năng cơ bản: người chơi với người, người chơi với máy và máy chơi với máy. Trong phạm vi của môn học này, nhóm hướng tới việc xây dựng một ứng dụng có thể tính toán không gian nước đi và tìm ra nước đi hợp lý nhất. Nhóm thực hiện đề tài này sử dụng hai thuật toán được giới thiệu trong môn học là thuật toán cắt tỉa $\alpha - \beta$ và thuật toán minimax, ngoài ra nhóm bổ sung thuật toán negamax là cải tiến của minimax.

- Thuật toán mini-max: Tìm kiếm trạng thái tối ưu của bảng cờ vua trong không gian nước đi có thể.
- Thuật toán cắt tỉa $\alpha - \beta$: Loại bỏ các nhánh cây không cần thiết để giảm số lượng không gian nước đi cần duyệt.
- Thuật toán negamax: cải tiến của thuật toán minimax, trong đó sử dụng cùng một hàm để tính toán cho người chơi max và min bằng cách đảo dấu

1.2.2 Kết quả mong muốn

Kết quả của việc kết hợp các thuật toán này với nhau là giúp máy có thể lựa chọn nước đi hợp lý để chống lại nước đi của đối thủ, đồng thời cải thiện thời gian phản hồi của máy nhằm tăng cường trải nghiệm cho người dùng.

Thông qua bài tập lớn của nhóm, các thành viên trong nhóm mong muốn có thể hiểu rõ được

bản chất của hai thuật toán đã được học là thuật toán cắt tỉa $\alpha - \beta$ và thuật toán mini-max. Đồng thời học được cách áp dụng hai thuật toán trên vào các ví dụ thực tiễn, cũng như so sánh chúng với khi sử dụng negamax

CHƯƠNG 2. PHƯƠNG PHÁP ÁP DỤNG

2.1 Các phương pháp liên quan

Trong thiết kế trò chơi cờ vua, có nhiều phương pháp và kỹ thuật quan trọng liên quan đến cả khía cạnh lập trình lẫn trải nghiệm người chơi. Dưới đây là một số phương pháp liên quan đến bài tập lớn:

- Thuật toán mini-max: Mục tiêu của thuật toán này là tìm kiếm nước đi tối ưu thông qua cây trạng thái(game tree) dựa trên nguyên tắc "tối ưu hóa lợi ích tối thiểu của đối thủ". Thuật toán sẽ đánh giá giá trị của từng nước đi và lựa chọn nước đi tối ưu
- Thuật toán nega-max: NegaMax là một phiên bản cải tiến của MiniMax dành cho các trò chơi đối kháng. Nó sử dụng một quan sát quan trọng: "Điểm số của Maximizer tại nút hiện tại bằng âm điểm số của Minimizer ở nút con kế tiếp." Do đó, NegaMax sử dụng cùng một hàm để tính toán cho cả người chơi Max và Min bằng cách đảo dấu (negative).
- Thuật toán cắt tỉa $\alpha - \beta$ (Alpha-Beta Pruning): Mục tiêu của thuật toán này là tối ưu hóa thuật toán mini-max bằng cách giảm số lượng nhánh cây được duyệt trên cây trạng thái bằng cách loại bỏ các nhánh của cây trạng thái mà không làm thay đổi kết quả tìm kiếm, qua đó giảm mức độ phức tạp của thuật toán mini-max. Qua đó, thuật toán giúp cải thiện độ phức tạp của thuật toán mini-max, đồng thời cải thiện trải nghiệm cho người dùng.
- Học máy trong cờ vua: Mục tiêu của thuật toán này là tạo ra đối thủ máy có khả năng học và cải thiện từ mỗi ván đấu. Thông qua mô hình học máy, máy sẽ tự động hóa quá trình cải thiện chiến thuật, ghi nhận và áp dụng những kinh nghiệm học được từ các ván đấu trước đó.
- Giao diện người dùng: Mục tiêu là tạo ra giao diện người chơi thân thiện và dễ sử dụng thông qua viết thiết kế giao diện đẹp, cung cấp các tính năng như lưu trò chơi, chia sẻ kết quả hay chức năng gợi ý nước đi,...
- Thuật toán đánh giá: Nhóm đánh giá kết quả của dự án thông qua thời gian phản hồi của máy và kết quả trong những trận đấu máy - người.

2.2 Phương pháp sẽ áp dụng

Với mục tiêu là xây dựng một mô hình trí tuệ nhân tạo thông qua các thuật toán đã được học trong môn học này, nhóm tập trung xây dựng trò chơi cờ vua với hai thuật toán cơ bản: Thuật toán $\alpha - \beta$ và thuật toán min-max. Đây là hai thuật toán kinh điển trong AI và vẫn đang được sử dụng trong các ứng dụng chơi cờ vua hiện nay. Ngoài ra nhóm còn xây dựng ba chế độ chơi riêng biệt bao gồm: Người chơi với người, người chơi với máy và máy chơi với máy.

2.3 Cấu trúc dữ liệu, thuật toán, dataset sử dụng đánh giá phương pháp

2.3.1 Cấu trúc dữ liệu, thuật toán

- Cấu trúc dữ liệu: Giao diện bàn cờ là một mảng 2 chiều các chuỗi tương ứng với tên các file ảnh (.png) của các quân cờ. Kí tự đầu tiên là đại diện cho màu cùa quân cờ (w: Trắng, b: Đen), kí tự thứ hai đại diên cho tên của quân cờ (p: Tốt, R: Xe, N: Mã, B: Tượng, Q: Hậu, K: Vua). Nước đi của người chơi được thể hiện qua lớp Move (được mô tả chi tiết trong mục 3.1.2).
- Thuật toán: Thuật toán được sử dụng là thuật toán Minimax kết hợp với cắt tỉa Alpha-Beta, và sử dụng thêm thuật toán negamax.

2.3.2 Phương pháp đánh giá

Các tiêu chí đánh giá performance: đánh giá thời gian AI ra quyết định đi nước cờ nhờ sự thay đổi của depth trong thuật toán, và cả tỉ lệ thắng của AI trong những lần chơi trực tiếp với con người.

CHƯƠNG 3. THỰC HIỆN XÂY CHƯƠNG TRÌNH

3.1 Xây dựng giao diện và luật chơi

3.1.1 Giao diện

```
import pygame as p
from chess import ChessEngine, SmartMoveFinder
WIDTH = HEIGHT = 512
DIMENSION = 8
MAX_FPS = 15
IMAGES = {}
def loadImages():
    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"), (SQ_SIZE, SQ_SIZE))
```

Giao diện được thực hiện bởi thư viện pygame. Kích thước bàn cờ được đặt là một hình vuông có kích thước 512x512 pixel, số khung hình tối đa trên 1 giây là 15. Hàm loadImage tải hình ảnh các quân cờ vào mảng IMAGES.

```
class GameState():
    def __init__(self):
        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"]
            ]
        self.moveNumber=0
        self.moveFunctions={'p':self.getPawnMoves,'R':self.getRookMoves,'N'
                            'B':self.getBishopMoves,'Q':self.getQueenMoves,'K':se
```

```
self.whiteToMove = True
self.moveLog=[]                    #luu tru cac nuoc da di
self.whiteKingLocation = (7, 4)
self.blackKingLocation = (0, 4)
self.checkMate = False
self.staleMate = False
self.enpassantPossible = ()
self.currentCastlingRight = CastleRights(True, True, True,
   True)
self.castleRightsLog =
   [CastleRights(self.currentCastlingRight.wks,
   self.currentCastlingRight.bks,
      self.currentCastlingRight.wqs,
         self.currentCastlingRight.bqs)]
```

Lớp GameState được định nghĩa dùng để mô tả trạng thái hiện tại của trò chơi bao gồm các thuộc tính:

- board: Thể hiện vị trí của các quân cờ hiện tại với giá trị của các phần tử tương ứng với tên các phần tử của trong mảng IMAGES chứa hình ảnh các quân cờ, phần tử có giá trị "—" là một ô không có quân cờ.
- moveNumber: Số nước đã đi
- moveFunctions: Từ điển định nghĩa các hàm chứa các nước đi hợp lệ với các quân cờ
- whiteToMove: Lưu lượt đánh hiện tại thuộc về bên nào
- moveLog: Lưu các nước cờ đã chơi
- whiteKingLocation: Vị trí hiện tại của vua trắng
- blackKingLocation: Vị trí hiện tại của vua đen
- checkMate: Kiểm tra chiếu hết
- staleMate: Kiểm tra hòa cờ
- enpassantPossible: Lưu các nước đi qua đường của các quân tốt
- currentCastlingRight: Kiểm tra quyền nhập thành trái phải của quân vua trắng và vua đen
- castleRightsLog: Lưu các trạng thái quyền nhập thành của các quân vua

```
def highlightSquares(screen, gs, validMoves, sqSelected):
    if sqSelected != ():
        r, c = sqSelected
        if gs.board[r][c][0] == ('w' if gs.whiteToMove else 'b'):
            s = p.Surface((SQ_SIZE, SQ_SIZE))
            s.set_alpha(100)
            s.fill(p.Color('blue'))
            screen.blit(s, (c*SQ_SIZE, r*SQ_SIZE))
            s.fill(p.Color('yellow'))
            for move in validMoves:
                if move.startRow == r and move.startCol == c:
                    screen.blit(s, (move.endCol * SQ_SIZE, move.endRow *
                        SQ_SIZE))
```

```
def drawBoard(screen):
    global colors
    colors = [p.Color("white"), p.Color("gray")]
    for r in range(DIMENSION):
        for c in range(DIMENSION):
            color = colors[(r + c) % 2]
            p.draw.rect(screen, color, p.Rect(c * SQ_SIZE, r * SQ_SIZE,
                SQ_SIZE, SQ_SIZE))


def drawPieces(screen, board):
    for r in range(DIMENSION):
        for c in range(DIMENSION):
            piece = board[r][c]
            if piece != "--":
                screen.blit(IMAGES[piece], p.Rect(c * SQ_SIZE, r *
                    SQ_SIZE, SQ_SIZE, SQ_SIZE))


def drawGameState(screen, gs, validMoves, sqSelected):
    drawBoard(screen)
    highlightSquares(screen, gs, validMoves, sqSelected)
    drawPieces(screen, gs.board)
```

Hàm drawBoard vẽ các ô trắng đen trên bàn cờ. Hàm drawPiece vẽ các quân cờ lên các ô tương ứng dựa trên ví trí các quân cờ được lưu trong board. Hàm highlightSquares làm nổi bật các nước đi hợp lệ của quân cờ mà người chơi đang chọn. Hàm drawGameState vẽ trạng thái hiện tại của trò chơi dựa trên 3 hàm trên.

```
  def draw_mode_selection_screen(screen):
      """
      Vẽ màn hình chọn chế độ.

      :param screen: pygame.Surface object, màn hình hiển thị.
      :return: Chế độ chơi (True cho "Chơi với người", False cho "Chơi
với máy").
      """
      screen.fill(p.Color("gray"))
      font = p.font.Font(None, 60)
      title_text = font.render("CHESS ARENA", True, p.Color("black"))
      title_rect = title_text.get_rect(center=(BOARD_WIDTH // 2 + 150,
200))
      screen.blit(title_text, title_rect)

      # Tạo các nút chọn chế độ
      button_font = p.font.Font(None, 36)
        button_text1 = button_font.render("Play With Human", True,
p.Color("black"))
         button_text2 = button_font.render("Play With AI", True,
p.Color("black"))

      button1_rect = p.Rect(BOARD_WIDTH // 2, 300, 300, 50)
      button2_rect = p.Rect(BOARD_WIDTH // 2, 400, 300, 50)

      # Vẽ lại các nút
       p.draw.rect(screen, p.Color("lightgray"), button1_rect)  # Nút
Play With Human
       p.draw.rect(screen, p.Color("lightgray"), button2_rect)  # Nút
Play With AI

      screen.blit(button_text1, button1_rect.move(50, 15))
      screen.blit(button_text2, button2_rect.move(80, 15))
      # Hiển thị hướng dẫn ở góc dưới trái
      instructions_font = p.font.SysFont("Arial", 22, False, False)
      instructions_text = "R: Restart the game | Z: Undo the previous
move | Home: Return to the main menu"
                                    instructions_surface
=
instructions_font.render(instructions_text, True, p.Color("black"))
      # Vẽ hướng dẫn lên màn hình tại vị trí góc dưới bên trái
```

```
        screen.blit(instructions_surface, (10, BOARD_HEIGHT - 30))
        instructions_font_name = p.font.SysFont("Arial", 26, False, False)
        instructions_name="Instructions:"
         instructions_surface_name=instructions_font_name.render(instruct
ions_name, True, p.Color("black"))
        screen.blit(instructions_surface_name,(10, BOARD_HEIGHT - 60))
        p.display.flip()

        # Chờ người chơi chọn chế độ
        while True:
            for event in p.event.get():
                if event.type == p.QUIT:
                    p.quit()
                    sys.exit()
                elif event.type == p.MOUSEBUTTONDOWN:
                    x, y = event.pos
                    if button1_rect.collidepoint(x, y):
                        return True  # Chọn chơi với người
                    elif button2_rect.collidepoint(x, y):
                        return False  # Chọn chơi với máy
```

Hàm drawBoard vẽ các ô trắng đen trên bàn cờ. Hàm drawPiece vẽ các quân cờ lên các ô tương ứng dựa trên ví trí các quân cờ được lưu trong board. Hàm highlightSquares làm nổi bật các nước đi hợp lệ của quân cờ mà người chơi đang chọn. Hàm drawGameState vẽ trạng thái hiện tại của trò chơi dựa trên 3 hàm trên.

```
def animateMove(move, screen, board, clock):
    global colors
    dR = move.endRow - move.startRow
    dC = move.endCol - move.startCol
    framesPerSquare = 10
    frameCount = (abs(dR) + abs(dC)) * framesPerSquare
    for frame in range(frameCount + 1):
        r, c = (move.startRow + dR * frame/frameCount, move.startCol +
            dC * frame/frameCount)
        drawBoard(screen)
        drawPieces(screen, board)
        color = colors[(move.endRow + move.endCol) % 2]
        endSquare = p.Rect(move.endCol*SQ_SIZE, move.endRow*SQ_SIZE,
            SQ_SIZE, SQ_SIZE)
        p.draw.rect(screen, color, endSquare)
        # if move.pieceCaptured != '--':
        #    screen.blit(IMAGES[move.pieceCaptured], endSquare)
        screen.blit(IMAGES[move.pieceMoved], p.Rect(c*SQ_SIZE,
            r*SQ_SIZE, SQ_SIZE, SQ_SIZE))
```

```
        p.display.flip()
        clock.tick(60)

def drawText(screen, text):
    font = p.font.SysFont("Helvitca", 32, True, False)
    textObject = font.render(text, 0, p.Color('Black'))
    textLocation = p.Rect(0, 0, WIDTH, HEIGHT).move(WIDTH/2 -
        textObject.get_width()/2, HEIGHT/2 - textObject.get_height()/2)
    screen.blit(textObject, textLocation)
    textObject = font.render(text, 0, p.Color("Black"))
    screen.blit(textObject, textLocation.move(2,2))
```

Hàm animateMove tạo hoạt ảnh cho các di chuyển của các quân cờ.

Hàm drawText hiển thị kết quả của ván chơi.

3.1.2 Luật chơi

```
class Move():

    ranksToRows={"1":7,"2":6,"3":5,"4":4,"5":3,"6":2,"7":1,"8":0}
    rowToRanks={v:k for k,v in ranksToRows.items()}
    fileToCols={"a":0,"b":1,"c":2,"d":3,"e":4,"f":5,"g":6,"h":7}
    colsToFiles={v:k for k,v in fileToCols.items()}
    def __init__(self,startsq,endsq,board,isEnpassantPossible=False,
        isCastleMove=False):
        self.startRow = startsq[0]
        self.endRow = endsq[0]
        self.startCol = startsq[1]
        self.endCol = endsq[1]
        self.pieceMoved = board[self.startRow][self.startCol]
        self.pieceCaptured = board[self.endRow][self.endCol]
        self.isPawnPromotion = (self.pieceMoved == "wp" and
            self.endRow == 0) or (self.pieceMoved == "bp" and
            self.endRow == 7)

        self.isEnpassantMove = isEnpassantPossible
        if self.isEnpassantMove:
            if self.pieceMoved == "bp":
                self.pieceCaptured = "wp"
            else:
                self.pieceCaptured = "bp"
        self.isCastleMove = isCastleMove
        self.moveID =
            self.startRow*1000+self.startCol*100+self.endRow*10+self.endCol
```

Lớp Move lưu các thông tin liên quan đến nước đi gồm vị trí bắt đầu của nước đi, ví trí kết thúc của nước đi, quân cờ thực hiện nước đi, quân cờ bắt được, kiểm tra xem nước đi đó có phải là nước đi dặc biệt: nhập thành, phong hậu, tốt đi qua đường.

```
class CastleRights():
    def __init__(self, wks, bks, wqs, bqs):
        self.wks = wks
```

```
        self.bks = bks
        self.wqs = wqs
        self.bqs = bqs
```

Lớp CastleRights lưu trạng thái quyền thực hiện nhập thành trái, phải của vua đen và vua trắng hiện tại

```
def inCheck(self):
    if self.whiteToMove:
        return self.squareUnderAttack(self.whiteKingLocation[0],
            self.whiteKingLocation[1])
    else:
        return self.squareUnderAttack(self.blackKingLocation[0],
            self.blackKingLocation[1])

def squareUnderAttack(self, r, c):
    self.whiteToMove = not self.whiteToMove
    oppMoves = self.getAllPossibelMoves()
    self.whiteToMove = not self.whiteToMove
    for move in oppMoves:
        if move.endRow == r and move.endCol == c:
            return True
    return False
```

Hàm squareUnderAttack kiểm tra xem một ví trị trên bàn cờ có bị tấn công bởi một quân cờ bất kì của đối phương hay không. Hàm inCheck kiểm tra vị trí của quân vua có bị tấn công bởi đối phương hay không.

```
    def getAllPossibelMoves(self):
        moves=[]
        for r in range(len(self.board)): #cow
            for c in range(len(self.board[r])): #row
                turn =self.board[r][c][0]    #while or black
                if (turn=='w' and self.whiteToMove) or(turn=='b'and not
                    self.whiteToMove):
                    piece=self.board[r][c][1]
                    self.moveFunctions[piece](r,c,moves)
        return moves
```

```
def getValidMoves(self):
    tmpEnpassantPossible = self.enpassantPossible
    tmpCastleRights = CastleRights(self.currentCastlingRight.wks,
        self.currentCastlingRight.bks,
                                self.currentCastlingRight.wqs,
                                    self.currentCastlingRight.bqs)
    moves = self.getAllPossibelMoves()
    if self.whiteToMove:
        self.getCastleMoves(self.whiteKingLocation[0],
            self.whiteKingLocation[1], moves)
    else:
        self.getCastleMoves(self.blackKingLocation[0],
            self.blackKingLocation[1], moves)
    for i in range(len(moves)-1, -1, -1):
        self.makeMove(moves[i])
        self.whiteToMove = not self.whiteToMove
        if self.inCheck():
            moves.remove(moves[i])
        self.whiteToMove = not self.whiteToMove
        self.undoMove()
    if len(moves) == 0:
        if self.inCheck():
            self.checkMate = True
        else:
            self.staleMate = True
    self.enpassantPossible = tmpEnpassantPossible
    self.currentCastlingRight = tmpCastleRights
    return moves
```

Hàm getAllPossibleMoves trả về tất cả các nước đi có thể của tất cả các quân cờ ở thời điểm hiện tại dựa trên từ điển moveFunctions các hàm xét các nước đi có thể của các quân cờ. Hàm getValidMoves kiểm tra trong tất cả các nước đi có thể, nước đi nào là hợp lệ. Một nước đi hợp lệ là nước đi không dẫn đến việc quân vua bị tấn công bởi một quân cờ bất kì của đối phương hoặc nước đi đó là một nước đi nhập thành hợp lệ.

```
def updateCastleRights(self, move):
    if move.pieceMoved == 'wK':
```

```
            self.currentCastlingRight.wks = False
            self.currentCastlingRight.wqs = False
        elif move.pieceMoved == 'bK':
            self.currentCastlingRight.bks = False
            self.currentCastlingRight.bqs = False
        elif move.pieceMoved == 'wR':
            if move.startRow == 7:
                if move.startCol == 0:
                    self.currentCastlingRight.wqs = False
                elif move.startCol == 7:
                    self.currentCastlingRight.wks = False
        elif move.pieceMoved == 'bR':
            if move.startRow == 0:
                if move.startCol == 0:
                    self.currentCastlingRight.bqs = False
                elif move.startCol == 7:
                    self.currentCastlingRight.bks = False
```

Hàm updateCastleRights cập nhật quyền nhập thành của quân vua sau mỗi nước đi. Nếu quân vua, quân xe đã di chuyển hoặc nước đi nhập thành dẫn đến quân vua bị tấn công thì quyền nhập thành đối với quân vua đó và cánh được xét chuyển thành False.

```
def main():
    #khoi tao man hinh choi c
    p.init()
    screen = p.display.set_mode((WIDTH, HEIGHT))
    clock = p.time.Clock()
    screen.fill(p.Color("white"))
    gs = ChessEngine.GameState()     #khoi tao trang thai ban dau
    animate = False
    moveMade=False #flag varible for when a move is made
    loadImages()            #tao mang chua anh cac quan c
    running = True
    sqSelected=()             #cac    c     c    click
    playerClicks=[]
    # lu  tr  thong tin click cua nguoi choi
    validMoves = gs.getValidMoves()
    gameOver = False
```

```
playerOne = True
playerTwo =False
```

Hàm main là hàm chạy chương trình. playerOne là bên cầm quân trắng, playTwo là bên cầm quân đen với giá trị True nếu do con người chơi, False do AI chơi. Người chơi click chuột vào quân cờ muốn đi và và click vào ô mà quân cờ di chuyển đến.

```
elif e.type==p.KEYDOWN:
          if e.key==p.K_z: # nhan z
             gs.undoMove()
             moveMade=True
             animate=False
             validMoves=gs.getValidMoves()
          if e.key==p.K_r:
             gs = ChessEngine.GameState()
             validMoves = gs.getValidMoves()
             sqSelected = ()
             playerClicks = []
             moveMade = False
             animate = False
```

Nhấn phím Z để thực hiện lùi lại một nước đi. Nhấn phím R để chơi lại.

```
 if gs.checkMate:
       gameOver =True
       if gs.whiteToMove:
          drawText(screen, "Black wins")
       else:
          drawText(screen, "White wins")
    elif gs.staleMate:
       gameOver =True
       drawText(screen, "Stalemate")
```

Khi xảy ra chiếu hết hoặc hòa cờ, màn hình hiển thị kết quả tương ứng của ván chơi.

3.2 Thuật toán tìm nước đi tiếp theo

3.2.1 Cách tính điểm không gian trạng thái

Điểm của không gian trạng thái được tỉnh dựa trên trạng thái, nếu checkmate thì sẽ là 10000 điểm, nếu stablemate là 0 điểm, còn lại được tính dựa trên các thành phần như sau

a. *Tính điểm quân cờ*

Điểm của các quân cờ là tổng điểm của tất cả các quân cờ ở trên bàn cờ với các điểm của từng loại quân cờ cụ thể như sau:

- Vua: 0 điểm (do Khi hết cờ thì điểm sẽ là 10000)
- Hậu: 900 điểm
- Xe: 500 điểm
- Tượng: 330 điểm
- Mã: 320 điểm
- Tốt: 100

b. *Tính điểm dựa trên vị trí các quân cờ*

Ngoài việc sử dụng điểm của các quân cờ, điểm của không gian trạng thái còn được tính dựa trên vị trí của các quân cờ.

Cụ thể một ván cờ được chia thành 3 giai đoạn:

- Khai cuộc: Không xét đến điểm dựa trên vị trí của các quân cờ
- Trung cuộc: Điểm dựa trên vị của các quân cờ đều được tính
- Tàn cuộc: Chỉ tính điểm dựa trên vị trí của quân tướng và quân tốt, tướng càng vào góc thì điểm càng thấp

3.2.2 Thuật toán Minimax

Thuật toán minimax là một thuật toán trong lĩnh vực trò chơi và trí tuệ nhân tạo được sử dụng để đưa ra quyết định trong các trò chơi tuyến tính có hai người chơi hoặc đối thủ. Mục tiêu của thuật toán là tối ưu hóa lợi ích của người chơi và tối thiểu hóa lợi ích của đối thủ.

Hàm thuật toán MiniMax sử dụng như sau

```
def findMoveMiniMax(gs, validMoves, depth, whiteToMove):
    """
    Hàm tìm điểm số của một nước đi bằng thuật toán Minimax.

    Parameters:
    gs (GameState): Trạng thái hiện tại của bàn cờ.
```

```
    validMoves (list): Danh sách các nước đi hợp lệ.
    depth (int): Độ sâu tìm kiếm.
    whiteToMove (bool): True nếu là lượt đi của quân trắng, False nếu là lượt đi
của quân đen.

    Returns:
    int: Điểm số của nước đi.
    """
    global nextMove, moveCounter
    moveCounter += 1
    random.shuffle(validMoves)
    if depth == 0:
        return scoreBoard(gs)

    if whiteToMove:
        maxScore = -CHECKMATE
        for move in validMoves:
            moveCounter += 1
            gs.makeMove(move)
            tmpCheckMate = gs.checkMate
            tmpStaleMate = gs.staleMate
            nextMoves = gs.getValidMoves()
            score = findMoveMiniMax(gs, nextMoves, depth - 1, False)
            if score > maxScore:
                maxScore = score
                if depth == DEPTH:
                    nextMove = move
            gs.checkMate = tmpCheckMate
            gs.staleMate = tmpStaleMate
            gs.undoMove()
        return maxScore
    else:
        minScore = CHECKMATE
        for move in validMoves:
            gs.makeMove(move)
```

```
        tmpCheckMate = gs.checkMate
        tmpStaleMate = gs.staleMate
        nextMoves = gs.getValidMoves()
        score = findMoveMiniMax(gs, nextMoves, depth - 1, True)
        if score < minScore:
            minScore = score
            if depth == DEPTH:
                nextMove = move
        gs.checkMate = tmpCheckMate
        gs.staleMate = tmpStaleMate
        gs.undoMove()
    return minScore
```

3.2.3 Thuật toán Negamax

Thuật toán Negamax là một biến thể của thuật toán Minimax.Negamax được thiết kế để giảm độ phức tạp trong mã nguồn và làm cho mã nguồn trở nên ngắn gọn hơn. Ý tưởng cơ bản của Negamax giống như thuật toán Minimax, tuy nhiên, nó sử dụng một kỹ thuật tối ưu hóa để giảm code.

3.2.4 Cắt tỉa Alpha-Beta

Cắt tỉa Alpha-Beta là một cải tiến của Minimax(cũng như Negamax) để giảm số lượng nút cần kiểm tra, tối ưu hóa thuật toán Minimax trong trò chơi hai người có lượt chơi xen kẽ.

Giả mã của thuật toán Negamax kết hợp với cắt tỉa alpha beta trong BTL như sau:

```
def findMoveNegaMax(gs, validMoves, depth, alpha, beta, turnMultiplier):
    """
    Hàm tìm điểm số của một nước đi bằng thuật toán NegaMax.

    Parameters:
    gs (GameState): Trạng thái hiện tại của bàn cờ.
    validMoves (list): Danh sách các nước đi hợp lệ.
    depth (int): Độ sâu tìm kiếm.
    alpha (int): Giá trị alpha.
    beta (int): Giá trị beta.
    turnMultiplier (int): Hệ số nhân cho điểm số.

    Returns:
    int: Điểm số của nước đi.
    """
    global nextMove, moveCounter
    random.shuffle(validMoves)
    if depth == 0:
        return turnMultiplier * scoreBoard(gs)

    maxScore = - CHECKMATE
    for move in validMoves:
        moveCounter += 1
        gs.makeMove(move)
        tmpCheckMate = gs.checkMate
        tmpStaleMate = gs.staleMate
        nextMoves = gs.getValidMoves()
        if len(nextMoves) == 0:
            if gs.checkMate:
                score = turnMultiplier * CHECKMATE
            elif gs.staleMate:
                score = STABLEMATE
        else:
            score = -findMoveNegaMax(gs, nextMoves, depth - 1, -beta, -alpha, -
turnMultiplier)
        if score > maxScore:
            maxScore = score
```

```
            if depth == DEPTH:
                nextMove = move
        gs.checkMate = tmpCheckMate
        gs.staleMate = tmpStaleMate
        gs.undoMove()
        if maxScore > alpha:
            alpha = maxScore
        if alpha >= beta:
            break
    return maxScore

def findBestMoveNegaMax(gs, validMoves):
    """
    Hàm tìm nước đi tốt nhất cho một bên bằng thuật toán Negamax.

    Parameters:
    gs (GameState): Trạng thái hiện tại của bàn cờ.
    validMoves (list): Danh sách các nước đi hợp lệ.

    Returns:
    tuple: Tọa độ của quân cờ được di chuyển và tọa độ đích của nước đi.
    """
    global nextMove, moveCounter, moveTime
    start_time = time.time()
    nextMove = None
    random.shuffle(validMoves)
    findMoveNegaMax(gs, validMoves, DEPTH, -CHECKMATE, CHECKMATE, 1 if
gs.whiteToMove else -1)
    end_time = time.time()
    moveTime = (end_time - start_time) * 1000
    return nextMove
```

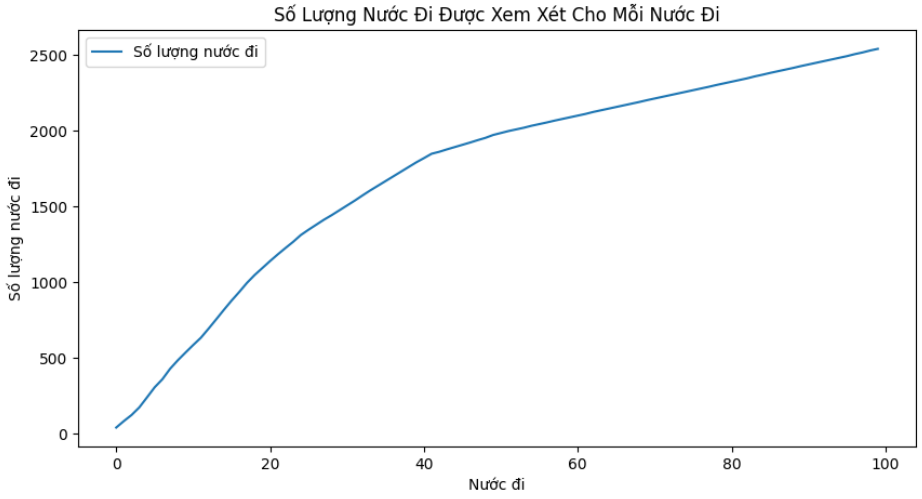
CHƯƠNG 4. KẾT LUẬN

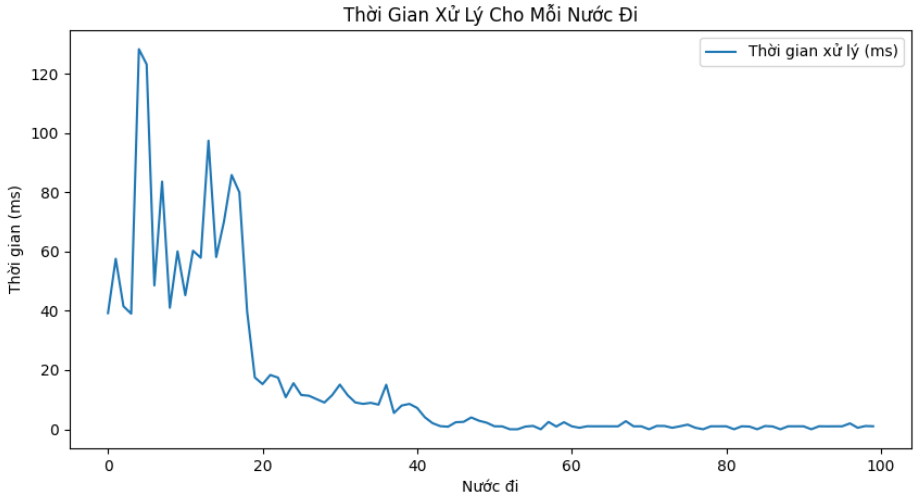
4.1 Kết quả và so sánh với mục tiêu đặt ra

Nhóm thử nghiệm trên cùng 1 người chơi thì thấy, với depth = 1, depth = 2 thì với thuật toán minimax hay negamax, cắt tỉa alpha beta máy chưa thể đánh thắng người được do nước đi tính toán còn ít. Nhưng với depth = 3 thì máy có thể chơi thắng người được với thời gian chờ cho máy ra quyết định hợp lí. Còn với depth = 4 thì chỉ cần đi được vài nước, không gian trang thái lớn lên, thời gian tính toán lâu hơn thậm chí có thể phải đợi 1 tiếng cho 1 nước đi nên nhóm không đưa vào đánh giá (hình 4.1, hình 4.2).

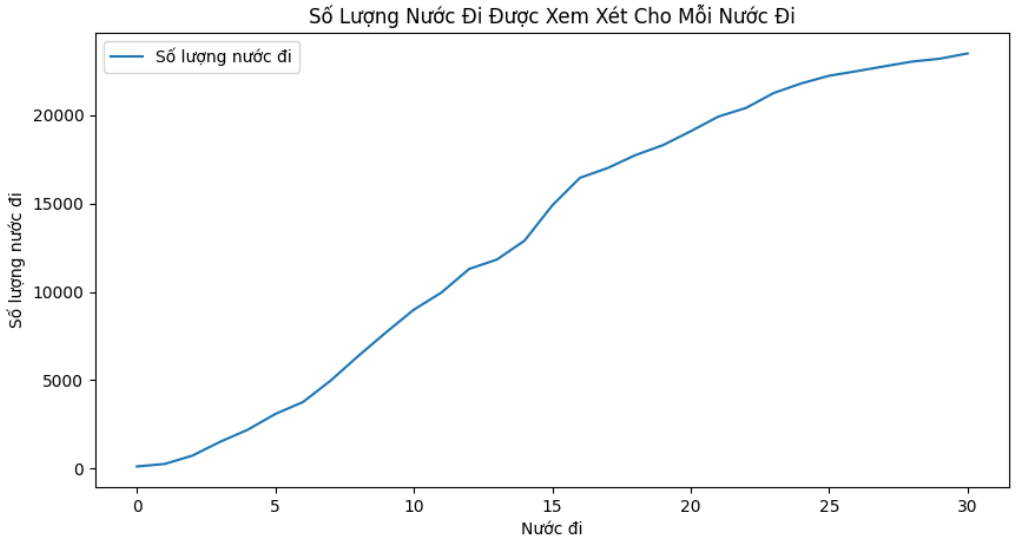
Thông số xét đối với quân đen (quân trắng tương tự)

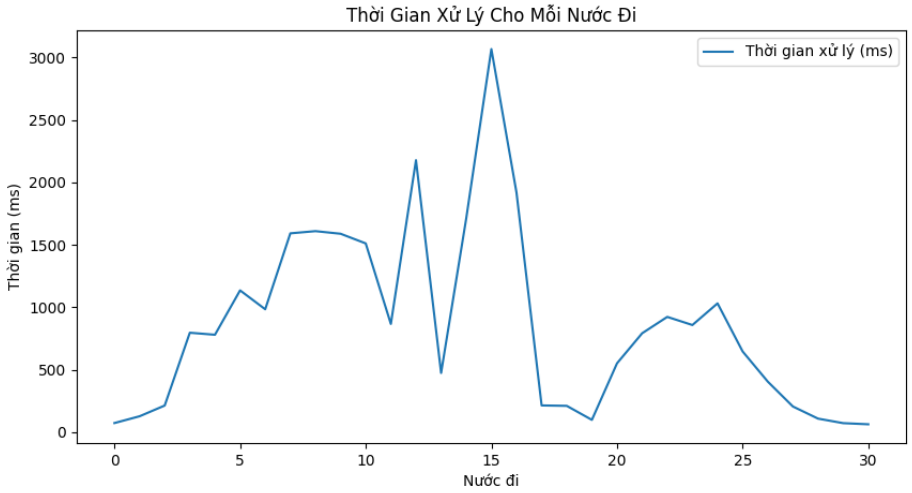
4.1.1 Depth=1:



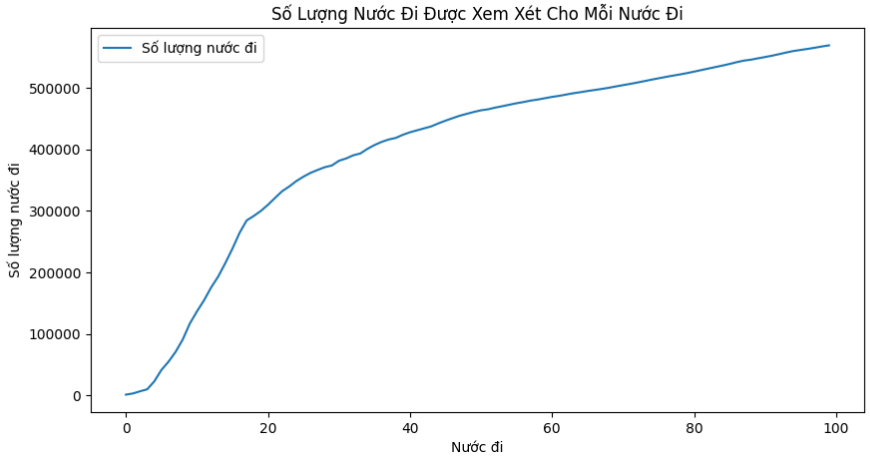


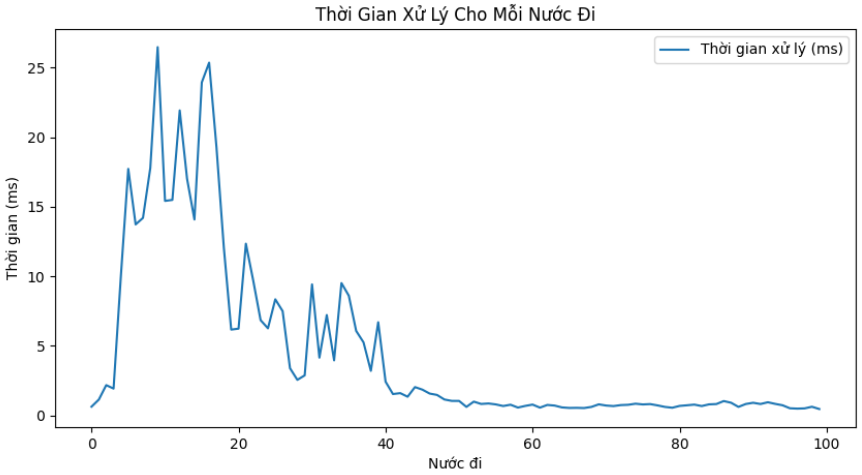
4.1.2 Depth=2:

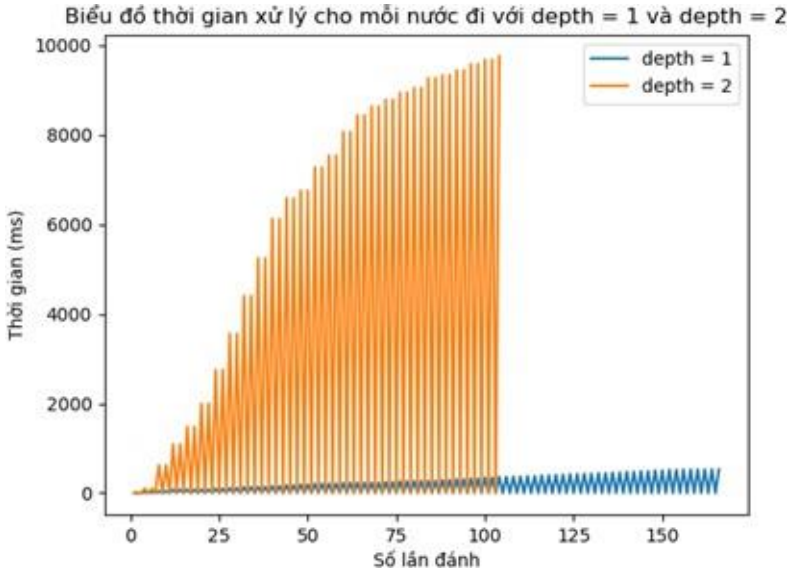




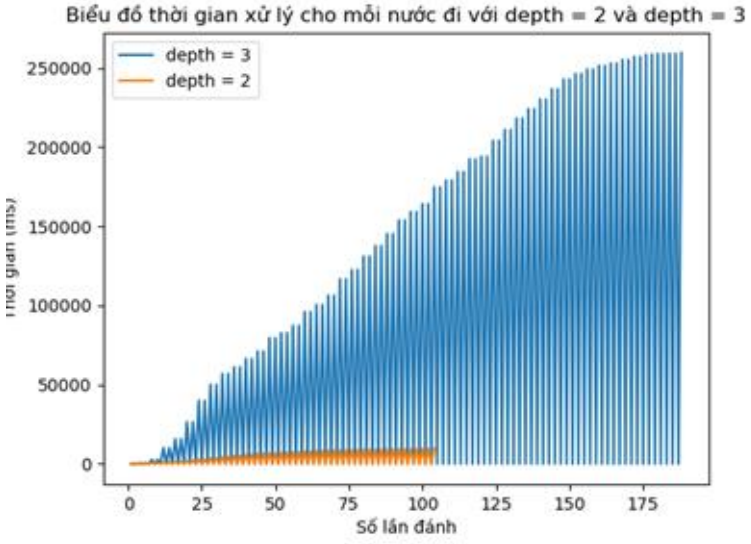
4.1.3. Depth =3:







Hình 4.1: Biểu đồ thời gian xử lý cho mỗi nước đi với depth = 1 và depth = 2



Hình 4.2: Biểu đồ thời gian xử lý cho mỗi nước đi với depth = 2 và depth = 3

Mục tiêu đặt ra là nhóm có thể xây dựng được một AI chơi cờ vua có thể thắng người. Điều này được thực hiện khi sử dụng thuật toán cắt tỉa alpha-beta để chơi cờ vua với depth = 3. Nhưng phần mềm chạy còn chưa nhanh như nhóm kì vọng, suy nghĩ trung bình khoảng 15s cho một nước đi với depth = 3 để đánh thắng người, vì thế cần phải cải tiến sâu hơn.

4.2 Hướng phát triển

Các hướng nghiên cứu tương lai tập trung vào:

- Áp dụng học máy vào heuristic: sử dụng Machine Learning để phát triển hàm đánh giá mạnh mẽ, có khả năng thích ứng với nhiều tình huống trò chơi.
- Cải thiện thuật toán Minimax và Negamax: Để cải thiện tốc độ của AI khi đánh, hiện tại khi depth=3 thì phần mềm suy nghĩ trung bình mất khoảng 15s, mục tiêu cải thiện khoảng 5s.
- Sử dụng Reinforcement learning để training AI chơi cờ, cho AI tự học hỏi từ môi trường cờ.
- Cải thiện giao diện để tăng trải nghiệm người dùng.

CHƯƠNG 5. THÔNG TIN KHÁC

5.1 Phân công công việc

Tuần	Công việc	Phụ trách
5	Chọn chủ đề + tìm hiểu phương pháp AI Làm báo cáo về chủ đề + phương pháp dự kiến	Cả nhóm Cả nhóm
6	Tìm hiểu thuật toán minimax, cắt tia alpha beta Thiết kế giao diện game, xác định rõ các chức năng chính của game	Cả nhóm An, Hiếu, Hùng
7	Tìm hiểu thêm một vài phương pháp, thuật toán cho cờ vua AI Hoàn thiện thiết kế Bắt đầu code giao diện và xây dựng logic cho game	My An, Hoàng Hùng, Hoàng, Hiếu
8	Hoàn thiện logic game Phát triển AI, tạo hành đánh giá bảng điểm Bắt đầu triển khai thuật toán Minimax	Hoàng, Hùng My, Hoàng
9	Hoàn thiện thuật toán Minimax và tối ưu Tích hợp AI vào Loop game, cho phép AI đưa ra các nước đi tự động Thử nghiệm nhanh và kiểm tra hoạt động của AI	My Hùng Hoàng
10	Viết proposal cho game Kiểm thử và hoàn thiện tất cả các tính năng logic của trò chơi	Hiếu Hoàng
11-12	Đánh giá toàn diện và tối ưu hóa bằng thuật toán căt tia Alpha-beta Đánh giá việc tôi ưu hóa thuật toán trong game	My Hùng, An
13-14	Đánh giá toàn diện và tối ưu hóa bằng thuật toán cắt tia Negamax Đánh giá việc tối ưu hóa thuật toán trong game	Hoàng My
	Viết file readme cho mã nguồn trên github	An
	Viết báo cáo bài tập lớn	Cả nhóm

TÀI LIỆU THAM KHẢO

[1] Eddie Sharick, *Creating a Chess Engine in Python*. Available: https://www.youtube.com/playlist?list=PLBwF487qi8MGU81nDGaeNE1EnNEPYWKY_ (visited on 9/10/2023).

[2] PSG.TS.Nguyễn Thanh Hương, *Slide nhập môn trí tuệ nhân tạo*. Trường Công nghệ Thông tin & Truyền thông - Đại học Bách Khoa Hà Nội