TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO:

HỌC PHẦN NGÔN NGỮ LẬP TRÌNH PYTHON

Đề tài:

Xây dựng ứng dụng game cờ caro bằng thư viện pygame

Họ và tên sinh viên: Huỳnh Khánh Duy

Mã số sinh viên: 3120410088

Lớp: DCT12010

Khoa: Công nghệ thông tin

Giảng viên hướng dẫn: Trịnh Tấn Đạt

Thành phố Hồ Chí Minh - Tháng 5/2022

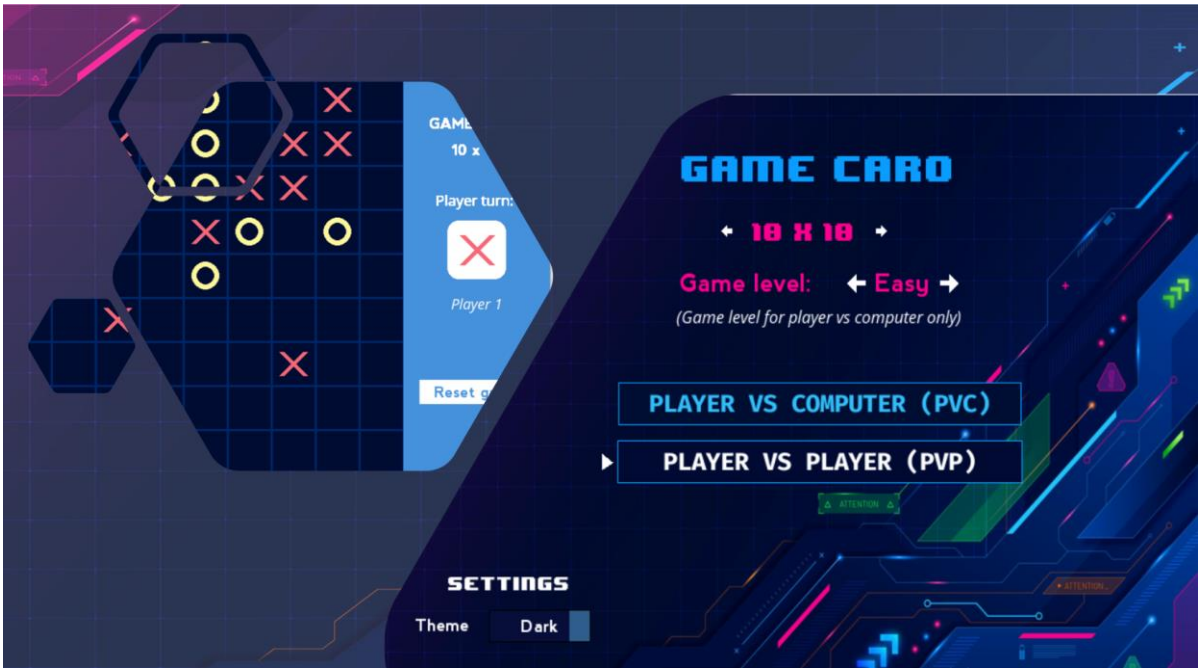
LỜI MỞ ĐẦU

Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ mã nguồn mở, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Thêm vào đó, python còn có rất là nhiều framework, liabry hỗ trợ người dùng như: Django, Flask, Tensorflow, OpenCV,… giúp cho việc lập trình trở nên dễ dàng và nhanh chóng hơn bao giờ hết.

Cờ ca-rô (hay sọc ca-rô) là một trò chơi dân gian. Cờ ca-rô trong tiếng Triều Tiên là omok và trong tiếng Nhật là gomoku narabe, tiếng Anh sử dụng lại tiếng Nhật, gọi là gomoku (hay Five in a Row). Ngoài ra cờ caro còn có 1 phiên bản nhỏ của cờ caro có số dòng và cột là 3 (3x3) gọi là Tic Tac Toe. Với quy luật vô cùng đơn giản, sẽ có hai người chơi đối kháng nhau. Nếu 1 trong hai đánh được 3 ô liên tiếp (đối với Tic Tac Toe) và 5 ô liên tiếp (đối với caro nhiều hơn 3 ô) nối thành 1 đường thẳng thì người chơi đó sẽ chiến thắng.

Trong đề tài này, ta sẽ sử dụng pygame (là một library của python đa nền tảng được thiết kế để viết trò chơi điện tử) và áp dụng các tính chất của trò chơi caro để xây dựng game cờ caro trên máy tính. Ngoài ra, đề tài còn sử dụng thuật toán minimax để lựa chọn bước đi kế tiếp phù hợp để lựa chọn bước đi trong game.

Link đồ án: https://github.com/huykhaduy/PythonCaroGame

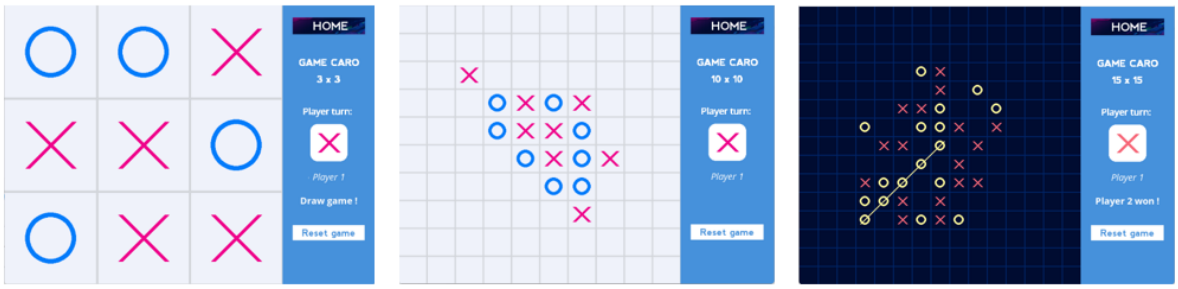


I. GIỚI THIỆU VỀ CHƯƠNG TRÌNH

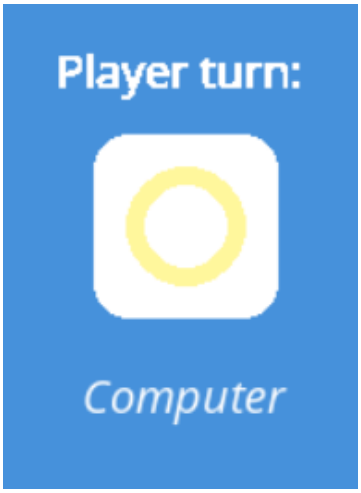
- Các chức năng của game:

+ Chọn kích cỡ của game: 3x3, 10x10, 15x15

3x3 10x10 15x15

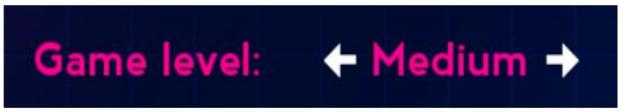


+ Chọn chế độ game: Người và người (PVP), Người và máy (PVC), hiển thị lượt chơi



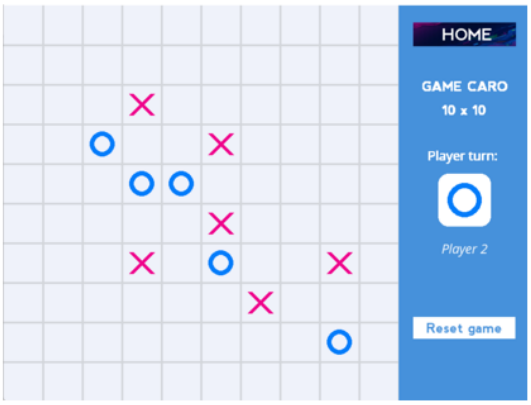
+ Chọn độ khó khi chơi với máy: Dễ (Easy), Trung bình (Medium) và Khó (Hard)

```
def evalMove(self, main_board):
    # print(self.aiLevel)
    if self.aiLevel == 1:
        # print("Easy")
        return self.easyLevel(main_board)
    elif self.aiLevel == 2:
        # print("Medium")
        return self.mediumLevel(main_board)
    else:
        # print("Hard")
        return self.hardLevel(main_board)

def easyLevel(self, main_board):...

def mediumLevel(self, main_board):...

def hardLevel(self, main_board):...
```



+ Chọn giao diện cho bàn cờ: chế độ sáng và tối.

Light Theme



Dark Theme

- Các bước sử dụng chương trình:

    + Chọn kích cỡ của bàn cờ caro

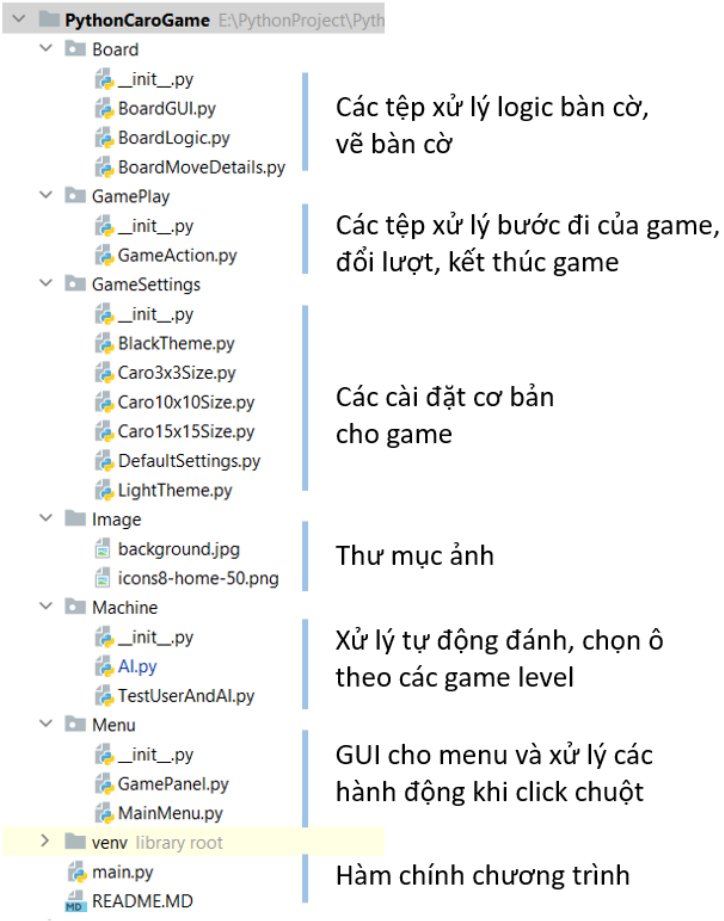
    + Chọn độ khó của trò chơi (nếu chơi với máy)

    + Chọn màu nền của game

    + Chọn chế độ muốn chơi (người hoặc máy)

- Cấu trúc các file trong chương trình:

Cấu trúc project



II. XÂY DỰNG CHƯƠNG TRÌNH

Đầu tiên, ta phải thực hiện xây dựng các lớp logic cơ bản cho chương trình. Các lớp này sẽ thực hiện các thao tác như tạo biến chứa bàn cờ, chứa thông tin các bước đi, kiểm tra chiến thắng hoặc hòa, lấy thông tin các ô trống, chạy thuật toán minimax… Sau đó sẽ tiến hành thực hiện vẽ bàn cờ, đánh dấu các bước đi của người chơi, xây dựng menu,…

1. Lớp BoardMoveDetails: thông tin bước đi trong bàn cờ

Trong mỗi bước đi của người chơi, ta cần phải tạo 1 class để chứa thông tin đó. Từ đó, giúp cho việc xác định thông tin bước đi của người chơi dễ dàng hơn.

- Các thuộc tính cơ bản:

+ self.playerId: id của người chơi

+ self.position: một tuple chứa dòng và cột

2. Lớp BasicBoardLogic: cài đặt logic bàn cờ caro

Tạo ra lớp BoardLogic để chứa các thông tin của bàn cờ, các hàm xử lý cơ bản của bàn cờ

- Các thuộc tính cơ bản:

+ self.board_row: số dòng của bảng

+ self.board_col: số cột của bảng

+ self. number_score_to_win: số điểm để thắng (đối với tictactoe là 3, còn lại là 5)

+ self.square_size: kích thước của mỗi ô hình vuông

+ self._squares: là một mảng 2 chiều dùng để chứa bàn cờ caro, có giá trị 0 đối với các ô trống và có giá trị 1 hay 2 đối với người chơi 1 hoặc 2.

+ self._number_of_turn: là tổng số lượt đã đi của hai người chơi

+ self._listMarked: là một list chứa danh sách các bước (lớp BoardTurnDetails)

+ self._winningLine: là một tuple chứa đoạn thẳng gồm điểm bắt đầu và điểm kết thúc

- Các hàm xử lý:

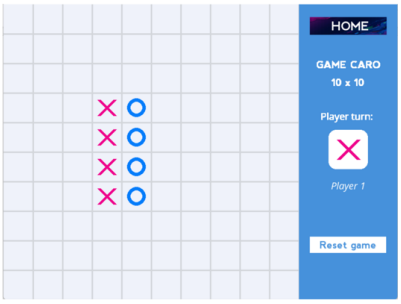
+ self.markSquare (playerId, row, col): thực hiện đánh dấu bước đi của người chơi có playerId tại vị trí dòng row và cột col. Khi đó, chương trình sẽ thực hiện thay đổi giá trị từ 0 của self._squares[row][col] thành playerId, đồng thời tổng lượt đã đánh lên 1 đơn vị. Sau đó thêm thông tin bước đi vào self._listMarked.

```
# Mark squares action
def markSquare(self, playerId, row, col):
    self._squares[row][col] = playerId
    self._number_of_turn += 1
    self._listMarked.append(BoardTurnDetails(playerId, row, col))
```

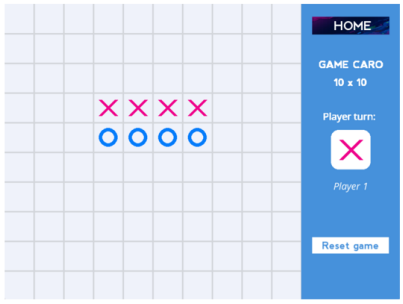
+ self.checkLineFromPos(playerId, row, col, changeRow, changeCol): thực hiện tìm ra đường thẳng liên tiếp của playerId từ vị trí dòng row cột col. Biến changeRow, changeCol dùng để xác định hướng đi của đường thẳng, tùy theo sự thay đổi của dòng và cột mà ta có thể chia ra 4 trường hợp như bên dưới.

(changeRow, changeCol)

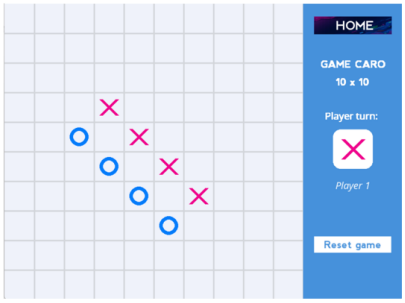
(1, 0): dòng thay đổi, cột đứng yên (theo chiều dọc)



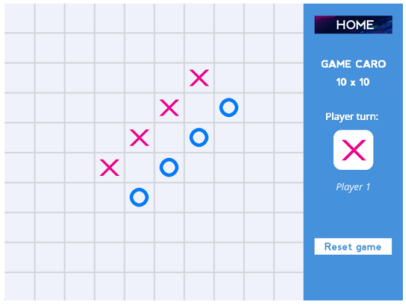
(0, 1): dòng đứng yên, cột thay đổi (theo chiều ngang)



(1, 1): dòng, cột cùng thay đổi (theo đường chéo chính)



(1, -1): dòng, cột thay đổi ngược nhau(theo đường chéo phụ)




```
def checkLineFromPos(self, playerId, row, col, changeRow, changeCol):
    # Get before squares from pos
    tempRow = row - changeRow
    tempCol = col - changeCol
    count = 1
    startPoint = (row, col)
    while self.isValidateRowCol(tempRow, tempCol) and self._squares[tempRow][tempCol] == playerId:
        count += 1
        startPoint = (tempRow, tempCol)
        tempRow -= changeRow
        tempCol -= changeCol

    # Get after squares from pos
    tempRow = row + changeRow
    tempCol = col + changeCol
    endPoint = (row, col)
    while self.isValidateRowCol(tempRow, tempCol) and self._squares[tempRow][tempCol] == playerId:
        count += 1
        endPoint = (tempRow, tempCol)
        tempRow += changeRow
        tempCol += changeCol

    # If win, save startPoint and endPoint of winning line
    isWinning = count >= self.number_score_to_win
    if isWinning:
        self._winningLine = (startPoint, endPoint)
    return isWinning, count
```

+ self.getWinningState(): hàm sẽ lấy thông tin bước đi cuối cùng trong self._listMarked và đưa nó vào hàm self.CheckLineFromPos để tìm các đường thẳng liên tiếp có thể dẫn đến chiến thắng của người chơi. Nếu có người chơi chiến thắng nó sẽ trả ra playerId, ngược lại trả về 0.

```
# Get board winning state, return playerId if it has won player else return 0
def getWinningState(self):
    if len(self._listMarked) == 0:
        return 0
    lastTurn = self._listMarked[-1]
    playerId = lastTurn.playerId
    row, col = lastTurn.position
    if self.checkLineFromPos(playerId, row, col, 1, 0)[0] \
            or self.checkLineFromPos(playerId, row, col, 0, 1)[0] \
            or self.checkLineFromPos(playerId, row, col, 1, 1)[0] \
            or self.checkLineFromPos(playerId, row, col, 1, -1)[0]:
        return playerId
    return 0
```

+ self.getEmptySquares(): hàm sẽ trả về các ô trong bảng có giá trị là 0, tức là chưa có người chơi đánh dấu.

```
# Get empty squares that didn't mark by player
def getEmptySquares(self):
    emptySqrs = []
    [emptySqrs.append((row, col)) for row in range(self.board_row)
     for col in range(self.board_col) if self._squares[row][col] == 0]
    return emptySqrs
```

+ self.isFull(): hàm sẽ kiểm tra nếu tổng số lượt đã đánh của hai người chơi có bằng với tổng số dòng * số cột của bảng hay không

```
def isFull(self):
    return self._number_of_turn >= self.board_row * self.board_col
```

+ self.isEmpty(): thực hiện kiểm tra số lượt đánh có bằng 0 hay không.

```
def isEmpty(self):
    return self._number_of_turn == 0
```

+ self.copyBoard(): sao chép các giá trị của bảng để cho AI thực hiện thuật toán Minimax

```
def copyBoard(self, other):
    self.board_col = other.board_col
    self.board_row = other.board_row
    self._number_of_turn = other.getNumberOfTurn()
    self._winningLine = copy.deepcopy(other.getWinningLine())
    self._listMarked = copy.deepcopy(other.getListMarked())
    self._squares = copy.deepcopy(other.getSquares())
```

+ self.getScoreOfPosition(self, playerId, row, col): Thực hiện tìm và đếm đường thẳng liên tục theo chiều ngang, dọc, chéo chính, chéo phụ từ điểm đó, rồi trả về giá trị lớn nhất.

```
def getScoreOfPosition(self, playerId, row, col):
    return max(self.checkLineFromPos(playerId, row, col, 1, 0)[1],
               self.checkLineFromPos(playerId, row, col, 0, 1)[1],
               self.checkLineFromPos(playerId, row, col, 1, 1)[1],
               self.checkLineFromPos(playerId, row, col, 1, -1)[1])
```

+ self.isValidateRowCol(row,col): kiểm tra dòng hay cột có hợp lệ hay không (lớn hơn 0 và bé hơn số dòng, cột của bảng)

```
# Check if row_index or col_index is invalid
def isValidateRowCol(self, row_index, col_index):
    if 0 <= row_index < self.board_row and 0 <= col_index < self.board_col:
        return True
    return False
```

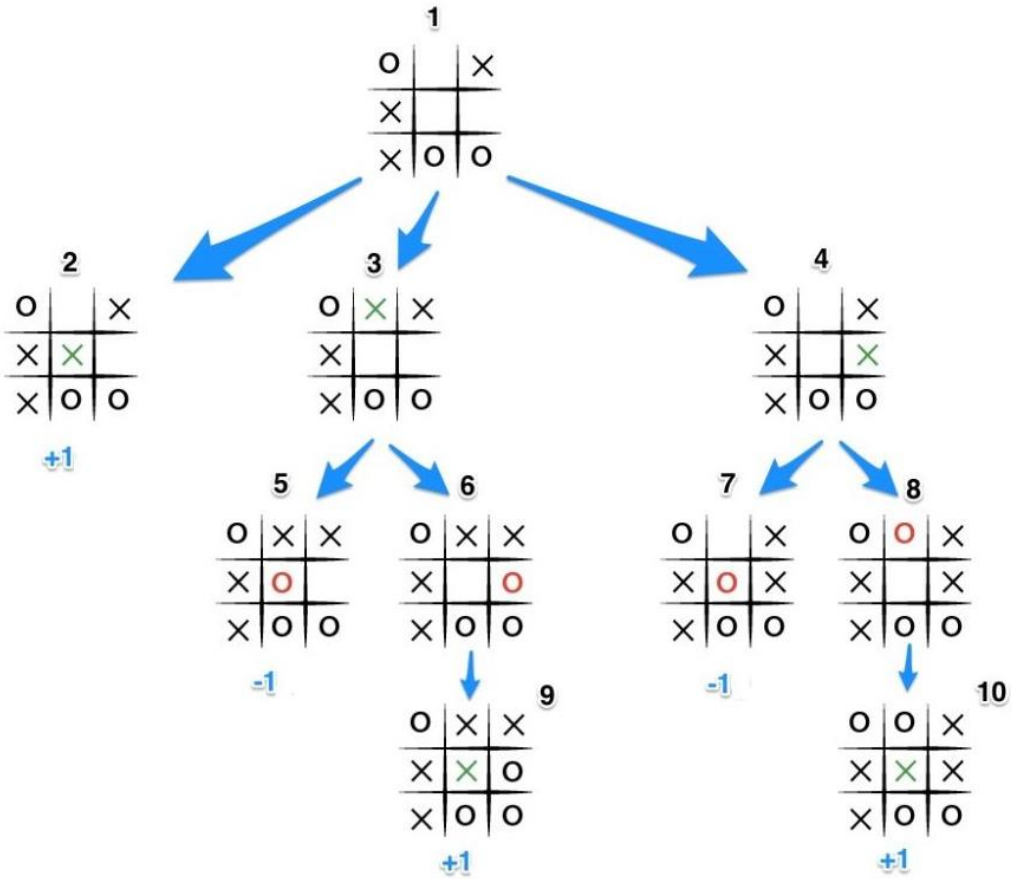
+ Ngoài ra, class AvancedBoardLogic đã kế thừa class BasicBoardLogic và thêm các hàm nâng cao như self.getScoreOfPosition (playerId, row, col) để tính điểm của ô nếu người chơi playerId nếu đánh tại dòng row, cột col

```
def getMostBenefitSqrs(self, selfPlayer, oppositePlayer):
    if self.isFull():
        return -1, -1
    emptySqr = self.getEmptySquares()
    max_point = 0
    next_mark = emptySqr[0]
    for row, col in emptySqr:
        self_point = self.getScoreOfPosition(selfPlayer, row, col)
        opposite_point = self.getScoreOfPosition(oppositePlayer, row, col)
        max_point = max(max_point, self_point, opposite_point)
        if opposite_point >= max_point:
            next_mark = (row, col)
        if self_point >= max_point:
            next_mark = (row, col)
    return next_mark
```

3. Lớp AI dựa trên thuật toán minimax

Minimax là giải thuật đệ quy nhằm lựa chọn bước đi kế tiếp trong một trò chơi có hai người bằng cách đệ quy tất cả trường hợp có thể xảy ra nhằm tìm trường hợp tốt nhất có thể đạt được.

Chìa khóa của thuật toán Minimax chính là lượt chơi qua lại giữa hai người chơi, khi mà cả hai đều mong muốn nước đi có điểm cao nhất. Điểm số của từng nước đi của ta được quyết định bởi đối thủ, người quyết định nước nào sẽ mang lại điểm số nhỏ nhất cho chúng ta. Ngược lại, điểm số của đối thủ lại được quyết định bởi chúng ta, người tìm cách tối ưu hóa điểm số. Nói ngắn gọn thì thuật toán Minimax trong trò chơi caro chính là thử hết các nước đi có thể cho đến khi trò chơi kết thúc.



Mã giả của thuật toán minimax:

```
Hàm minimax(bàn cờ, lượt chơi):
  Nếu tất cả ô đầy:
      Trả về 0, None

  Nếu người chơi chiến thắng (người chơi là max):
      Trả về 1, None

  Nếu máy chiến thắng (máy là min):
      Trả về -1, None

  Nếu là lượt chơi của máy:
      # Khởi tạo giá trị min vô cùng lớn
      Giá trị min = 100
      For các ô trống trong bảng:
          Thực hiện đánh dấu ô trống
          Thay đổi lượt chơi
          Giá trị trả về = Đệ qui với bảng đã đánh dấu ô trống
          Nếu giá trị trả về nhỏ hơn giá trị min thì thay thế giá trị min
      Trả về giá trị min và ô trống có giá trị tốt nhất

  Nếu là lượt chơi của người chơi:
      # Khởi tạo giá trị max vô cùng nhỏ
      Giá trị max = -100
      For các ô trống trong bảng:
          Thực hiện đánh dấu ô trống
```

```
        Thay đổi lượt chơi
        Giá trị trả về = Đệ qui với bảng đã đánh dấu ô trống
        Nếu giá trị trả về lớn hơn giá trị max thì thay thế giá trị max
    Trả về giá trị max và ô trống có giá trị tốt nhất
```

Từ giải thuật trên, ta có thể thấy thuật toán minimax khi áp dụng cho các bàn cờ cỡ lớn sẽ cực kì tốn nhiều chi phí cho việc sử dụng đệ qui tất cả các thành phần trong bảng. Cho nên trong đồ án này, sẽ chỉ sử dụng thuật toán minimax cho bàn cờ cỡ 3x3.

- Các thuộc tính của lớp AI:

+ self.aiLevel: mức độ khó của game (tương ứng với 1 – Easy, 2 – Medium, 3 – Hard)

+ self.aiPlayer: mã id của máy

+ self.userPlayer: mã id của người chơi

- Các hàm của lớp AI:

+ self.minimax(board, isMaximizing, depth):Hàm lựa chọn bước đi dựa trên thuật toán minimax. Tuy nhiên chi phí bộ nhớ của hàm sử dụng khá nhiều do việc tính toán các nhánh không cần thiết.

+ self.minimax_update(board, isMaximizing, alpha, beta, depth): Hàm lựa chọn bước đi dựa trên minimax đã sử dụng thuật toán cắt tỉa cây alpha beta để loại bỏ các nhánh không cần thiết.

```python
def minimax(self, board: AdvancedBoardLogic, isMaximizing, depth):
    if board.isFull() or depth == 0:
        return 0, None

    if board.getWinningState() == self.userPlayer:
        return 1, None

    if board.getWinningState() == self.aiPlayer:
        return -1, None

    if isMaximizing:
        maxEval = -100
        bestMove = None
        emptySqrs = board.getEmptySquares()
        for (row, col) in emptySqrs:
            tempBoard = copy.deepcopy(board)
            tempBoard.markSquare(self.userPlayer, row, col)
            myEval = self.minimax(tempBoard, False, depth - 1)[0]
            if myEval > maxEval:
                maxEval = myEval
                bestMove = (row, col)
        return maxEval, bestMove

    if not isMaximizing:
        minEval = 100
        bestMove = None
        emptySqrs = board.getEmptySquares()
        for (row, col) in emptySqrs:
            tempBoard = copy.deepcopy(board)
            tempBoard.markSquare(self.aiPlayer, row, col)
            myEval = self.minimax(tempBoard, True, depth - 1)[0]
            if myEval < minEval:
                minEval = myEval
                bestMove = (row, col)
        return minEval, bestMove
```

```python
def minimax_update(self, board, isMaximizing, alpha, beta, depth):
    if board.isFull() or depth == 0:
        return 0, None

    if board.getWinningState() == self.userPlayer:
        return 1, None

    if board.getWinningState() == self.aiPlayer:
        return -1, None

    if isMaximizing:
        maxEval = -100
        bestMove = None
        emptySqrs = board.getEmptySquares()
        for (row, col) in emptySqrs:
            tempBoard = copy.deepcopy(board)
            tempBoard.markSquare(self.userPlayer, row, col)
            myEval = self.minimax_update(tempBoard, False, alpha, beta, depth - 1)[0]
            if myEval > maxEval:
                maxEval = myEval
                bestMove = (row, col)
            alpha = max(alpha, myEval)
            if beta <= alpha:
                break
        return maxEval, bestMove

    if not isMaximizing:
        minEval = 100
        bestMove = None
        emptySqrs = board.getEmptySquares()
        for (row, col) in emptySqrs:
            tempBoard = copy.deepcopy(board)
            tempBoard.markSquare(self.aiPlayer, row, col)
            myEval = self.minimax_update(tempBoard, True, alpha, beta, depth - 1)[0]
            if myEval < minEval:
                minEval = myEval
                bestMove = (row, col)
            alpha = min(beta, myEval)
            if beta <= alpha:
                break
        return minEval, bestMove
```

+ self.random_Level(main_board): Trả về một ô ngẫu nhiên trong các ô trống trong bảng

```
def randomLevel(self, main_board):
    emptySqrs = main_board.getEmptySquares()
    move = emptySqrs[random.randint(0, len(emptySqrs) - 1)]
    return move
```

+ self.mostMove(main_board): Trả về ô có lợi nhất trong bảng

```
def mostMove(self, main_board):
    return main_board.getMostBenefitSqrs(self.aiPlayer, self.userPlayer)
```

+ self.easyLevel(main_board): Gọi hàm mostMove và trả về kết quả

```
def easyLevel(self, main_board):
    return self.mostMove(main_board)
```

+ self.mediumLevel(main_board): Nếu là 2 lượt chơi đầu sẽ gọi hàm random trả về kết quả. Ngược lại sẽ gọi minmax và tìm giá trị tốt nhất cho bảng

```
def mediumLevel(self, main_board):
    if main_board.getNumberOfTurn() < 2:
        move = self.randomLevel(main_board)
    else:
        myEval, move = self.minimax(main_board, False, 1000)
    return move
```

+ self.hardLevel(main_board): Gọi hàm minimax_update sử dụng thuật toán cắt tỉa cây để tối ưu tốc độ.

```
def hardLevel(self, main_board):
    myEval, move = self.minimax_update(main_board, False, -100, 100, 1000)
    return move
```

4. Xây dựng giao diện bằng pygame

Đầu tiên, ta phải import thư viện pygame. Sau đó khởi tạo pygame và cài đặt các thuộc tính như kích cỡ màn hình, title của game (các biến screen_width, screen_height được lưu ở DefaultSettings).

```
pygame.init()
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('Trò chơi cờ caro')
```

Ta tạo lớp BoardGUI trong package Board sau đó cài đặt các hàm như:

- Vẽ bảng game: vẽ các đường thẳng theo chiều dọc và ngang.

```
def drawBoardGames(self):
    self.screen.fill(self.theme.background_color)
    # Draw vertical lines
    x_index = self.square_size
    for col in range(self.board_col):
        pygame.draw.line(self.screen, self.theme.line_color, (x_index, 0), (x_index, board_height), self.size.line_width)
        x_index += self.square_size
    # Draw horizontal lines
    y_index = self.square_size
    for row in range(self.board_col):
        pygame.draw.line(self.screen, self.theme.line_color, (0, y_index), (board_width, y_index), self.size.line_width)
        y_index += self.square_size
```

- Vẽ bước đi của người chơi:

```
# Override BasicBoardLogic function
def markSquare(self, playerId, row, col):
    super().markSquare(playerId, row, col)
    self.drawPlayerMark(playerId, row, col)
    boardState = self.getWinningState()
    if boardState != 0:
        startPoint, endPoint = self._winningLine
        self.drawWinningLine(boardState, startPoint, endPoint)
    return boardState

def drawPlayerMark(self, playerId, row, col):
    # Draw cross shape
    if playerId == cross_player:
        # Main cross line
        main_cross_start = (col * self.square_size + self.size.offset, row * self.square_size + self.size.offset)
        main_cross_end = (
            col * self.square_size + self.square_size - self.size.offset, row * self.square_size + self.square_size - self.size.offset)
        pygame.draw.line(self.screen, self.theme.cross_color, main_cross_start, main_cross_end, self.size.cross_width)
        # Extra cross line
        extra_cross_start = (col * self.square_size + self.square_size - self.size.offset, row * self.square_size + self.size.offset)
        extra_cross_end = (col * self.square_size + self.size.offset, row * self.square_size + self.square_size - self.size.offset)
        pygame.draw.line(self.screen, self.theme.cross_color, extra_cross_start, extra_cross_end, self.size.cross_width)
    # Draw circle shape
    elif playerId == circle_player:
        center = (col * self.square_size + self.square_size // 2, row * self.square_size + self.square_size // 2)
        pygame.draw.circle(self.screen, self.theme.circle_color, center, self.size.circle_radius, self.size.circle_width)
```

- Vẽ đường chéo chiến thắng: theo chiều ngang, dọc, chéo chính và phụ

```
def drawWinningLine(self, playerId, startPoint, endPoint):
    # Get line color for player 1 or player 2
    color = None
    if playerId == cross_player:
        color = self.theme.cross_color
    elif playerId == circle_player:
        color = self.theme.circle_color
    else:
        return
    # Draw lines main diagonal
    if startPoint[0] - endPoint[0] == startPoint[1] - endPoint[1]:
        start = (startPoint[1] * self.square_size + self.size.offset, startPoint[0] * self.square_size + self.size.offset)
        end = (endPoint[1] * self.square_size + self.square_size - self.size.offset,
               endPoint[0] * self.square_size + self.square_size - self.size.offset)
        pygame.draw.line(self.screen, color, start, end, self.size.win_width)
    # Draw lines auxiliary diagonal
    elif startPoint[0] - endPoint[0] == -(startPoint[1] - endPoint[1]):
        start = (
            startPoint[1] * self.square_size + self.square_size - self.size.offset, startPoint[0] * self.square_size + self.size.offset)
        end = (endPoint[1] * self.square_size + self.size.offset, endPoint[0] * self.square_size + self.square_size - self.size.offset)
        pygame.draw.line(self.screen, color, start, end, self.size.win_width)
    # Draw line row
    elif startPoint[1] == endPoint[1]:
        start = (
            startPoint[1] * self.square_size + self.square_size // 2, startPoint[0] * self.square_size + self.size.offset)
        end = (endPoint[1] * self.square_size + self.square_size // 2,
               endPoint[0] * self.square_size + self.square_size - self.size.offset)
        pygame.draw.line(self.screen, color, start, end, self.size.win_width)
    # Draw line column
    else:
        start = (
            startPoint[1] * self.square_size + self.size.offset, startPoint[0] * self.square_size + self.square_size // 2)
        end = (endPoint[1] * self.square_size + self.square_size - self.size.offset,
               endPoint[0] * self.square_size + self.square_size // 2)
        pygame.draw.line(self.screen, color, start, end, self.size.win_width)
```

Sau đó, ta tạo lớp GameAction để quản lý các hành động, điều khiển game:

```
class GameAction:
    def __init__(self, screen, theme, menu, boardSize):
        self.screen = screen
        self.theme = theme
        self.row = boardSize
        self.col = boardSize
        self.board = BoardGUI(self.screen, theme, self.row, self.col)
        self.playerTurn = 1
        self.game_mode = mode_pvp
        self.is_running = True
        self.board.drawBoardGames()
        self.menu = menu
        self.panel = GamePanel(screen, theme, self.resetGame, self.menu, boardSize)
```

Các hàm xử lý cơ bản của GameAction:

```
# Thực hiện thao tác đánh cờ
def move(self, row, col):
    boardState = self.board.markSquare(self.playerTurn, row, col)
    if boardState != 0:
        self.panel.showWinningTitle("Player " + str(self.playerTurn) + " won !")
        self.panel.menu.draw(self.screen)
        self.is_running = False
    if self.board.isFull() and boardState == 0:
        self.is_running = False
        self.panel.showWinningTitle("Draw game !")
    self.nexTurn()

# Chuyển đến lượt tiếp theo
def nexTurn(self):
    self.playerTurn = self.playerTurn % 2 + 1

# Kiểm tra game kết thúc
def isOver(self):
    if self.board.getWinningState() != 0 or self.board.isFull():
        self.is_running = False
        return True
    return False

# Reset game
def resetGame(self):
    self.playerTurn = 1
    self.is_running = True
    self.board = BoardGUI(self.screen, self.theme, self.row, self.col)
    self.board.drawBoardGames()
    self.panel = GamePanel(self.screen, self.theme, self.resetGame, self.menu, self.row)
    self.panel.showWinningTitle(" " * 30)
```

Hai hàm cho chế độ chơi với người hay máy của GameAction

```
# Chơi với người
def runGamePVP(self):
    while True:
        events = pygame.event.get()
        self.panel.display(events, self.screen)
        for event in events:
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            self.panel.run(events, self.playerTurn, "Player " + str(self.playerTurn))
            if self.is_running and event.type == pygame.MOUSEBUTTONDOWN:
                pos = event.pos
                row = pos[1] // self.board.square_size
                col = pos[0] // self.board.square_size
                if self.board.isSquareEmpty(row, col):
                    self.move(row, col)
                pygame.display.update()
        pygame.display.update()
```

```
# Chơi với máy
def runGameAI(self, aiLevel=1):
    ai = AI(aiLevel)
    while True:
        events = pygame.event.get()
        self.panel.display(events, self.screen)
        for event in events:
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            if self.is_running and ai.userPlayer == self.playerTurn:
                self.panel.run(events, self.playerTurn, "Player " + str(self.playerTurn))
                if event.type == pygame.MOUSEBUTTONDOWN:
                    pos = event.pos
                    row = pos[1] // self.board.square_size
                    col = pos[0] // self.board.square_size
                    if self.board.isSquareEmpty(row, col):
                        self.move(row, col)
                pygame.display.update()

            if self.is_running and ai.aiPlayer == self.playerTurn:
                self.panel.run(events, self.playerTurn, "Computer")
                none_gui_board = AdvancedBoardLogic()
                none_gui_board.copyBoard(self.board)
                row, col = ai.evalMove(none_gui_board)
                self.move(row, col)
                pygame.display.update()
        pygame.display.update()
```

Tạo class Menu để chứa giao diện menu gồm:

1. Set background cho Menu

```
self.menuTheme = pygame_menu.Theme(
    background_color=pygame_menu.baseimage.BaseImage(image_path="./Image/background.jpg"),
    title_bar_style=pygame_menu.widgets.MENUBAR_STYLE_NONE,
    widget_alignment=pygame_menu.locals.ALIGN_CENTER,
    title=False,
    widget_selection_effect=pygame_menu.widgets.LeftArrowSelection(arrow_size=(50, 50)).set_background_color(
        (0, 10, 51))
)
```

2. Thiết lập cài đặt menu chính

```
self.menu = pygame_menu.Menu(
    height=screen_height,
    theme=self.menuTheme,
    title='',
    width=screen_width,
    center_content=False,
    mouse_motion_selection=True

)
```

3. Thêm tựa đề cho game và tạo selector cho kích cỡ game

```
self.menu.add.label("Game Caro",
                    font_name=pygame_menu.font.FONT_8BIT,
                    font_color=(11, 156, 255)
                    ).translate(0, 50)

items = [('3 x 3', 3),
         ('10 x 10', 10),
         ('15 x 15', 15)]
self.selector1 = self.menu.add.selector(
    title='',
    items=items,
    selection_effect=pygame_menu.widgets.NoneSelection(),
    cursor=pygame_menu.locals.CURSOR_HAND,
    font_color=(254, 0, 143),
    font_name=pygame_menu.font.FONT_8BIT,
    font_size=20,
    style=pygame_menu.widgets.SELECTOR_STYLE_FANCY,
    style_fancy_bgcolor=pygame_menu.themes.TRANSPARENT_COLOR,
    style_fancy_bordercolor=pygame_menu.themes.TRANSPARENT_COLOR,
    style_fancy_arrow_color=(255, 255, 255),
    style_fancy_arrow_margin=(15, 15, 0)
).translate(0, 80)
```

4. Thêm selector lựa chọn các mức độ trong game

```
levels = [("Easy", 1), ("Medium", 2), ("Hard", 3)]
self.selector2 = self.menu.add.selector(
    title='\t\tGame level:',
    items=levels,
    align=pygame_menu.locals.ALIGN_CENTER,
    selection_effect=pygame_menu.widgets.NoneSelection(),
    cursor=pygame_menu.locals.CURSOR_HAND,
    font_color=(254, 0, 143),
    font_name=pygame_menu.font.FONT_NEVIS,
    font_size=24,
    style=pygame_menu.widgets.SELECTOR_STYLE_FANCY,
    style_fancy_bgcolor=pygame_menu.themes.TRANSPARENT_COLOR,
    style_fancy_bordercolor=pygame_menu.themes.TRANSPARENT_COLOR,
    style_fancy_arrow_color=(255, 255, 255),
    # style_fancy_arrow_margin=(15, 15, 0)
).translate(0, 100)
```

5. Thêm button chọn lựa chơi với người hay chơi với máy

```
self.menu.add.button('PLAYER VS COMPUTER (PVC)', action=self.initPVCGame,
                     font_color=(51, 191, 251),
                     font_name=pygame_menu.font.FONT_FIRACODE_BOLD,
                     align=pygame_menu.locals.ALIGN_CENTER,
                     border_width=1,
                     font_size=24,
                     padding=(5, 30),
                     cursor=pygame_menu.locals.CURSOR_HAND,
                     border_color=(3, 160, 253),
                     background_color=(0, 14, 51),
                     ).translate(0, 150)
self.menu.add.button('PLAYER VS PLAYER (PVP)', action=self.initPVPGame,
                                    font_color=(253, 0, 143),
                                    font_name=pygame_menu.font.FONT_FIRACODE_BOLD,
                                    align=pygame_menu.locals.ALIGN_CENTER,
                                    border_width=1,
                                    cursor=pygame_menu.locals.CURSOR_HAND,
                                    font_size=24,
                                    padding=(5, 45),
                                    border_color=(3, 160, 253),
                                    background_color=(0, 14, 51)
                                    ).translate(0, 165)
```

6. Thêm switch chọn màu cho game

```
self.menu.add.label("Settings",
                    font_name=pygame_menu.font.FONT_8BIT,
                    font_color=(255, 255, 255),
                    align=pygame_menu.locals.ALIGN_LEFT,
                    font_size=16
                    ).translate(0, 250)
self.theme = self.menu.add.toggle_switch('Theme', False, width=100,
                                         font_name=pygame_menu.font.FONT_NEVIS,
                                         font_color=(255, 255, 255), padding=0,
                                         selection_effect=pygame_menu.widgets.NoneSelection(),
                                         align=pygame_menu.locals.ALIGN_LEFT,
                                         font_size=16, state_text=('Light', 'Dark'),
                                         slider_color=(48, 94, 140),
                                         state_color=((255, 255, 255), (8, 14, 58)),
                                         switch_margin=(20, 0),
                                         state_text_font_color=((8, 14, 58), (255, 255, 255)),
                                         switch_height=1.8,
                                         switch_border_width=1,
                                         cursor=pygame_menu.locals.CURSOR_HAND
                                         ).translate(30, 265)
```

Khi người dùng click chơi với máy hoặc chơi với người button đó sẽ gọi hàm self.initPVCGame() hoặc self.initPVPGame(). Trong các hàm đó sẽ lấy thông tin từ các selector, switch mà người dùng đã chọn mà khởi tạo game.

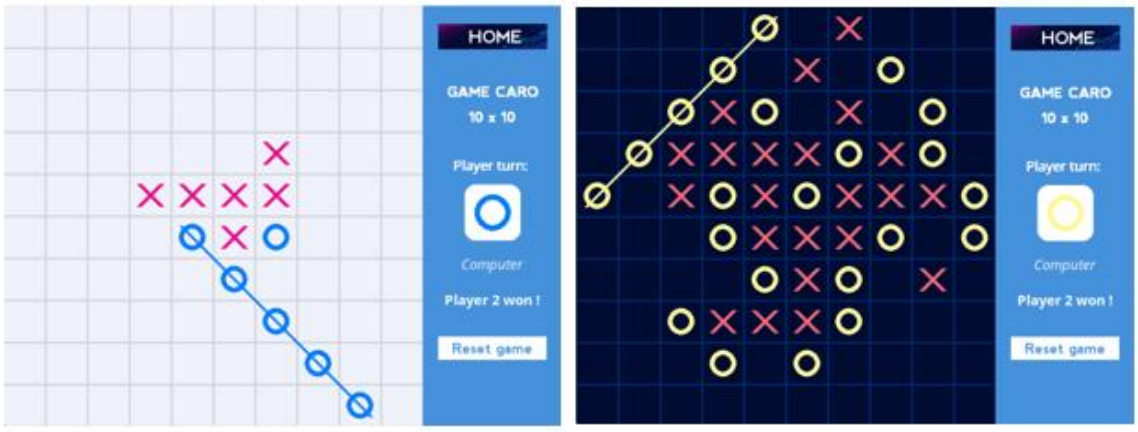
```
def initPVPGame(self):
    boardSize = self.selector1.get_value()[0][1]
    gameTheme = LightTheme
    if self.theme.get_value():
        gameTheme = BlackTheme
    game = GameAction(self.screen, gameTheme, self.loop, boardSize)
    game.runGamePVP()

def initPVCGame(self):
    boardSize = self.selector1.get_value()[0][1]
    aiLevel = self.selector2.get_value()[0][1]
    gameTheme = LightTheme
    if self.theme.get_value():
        gameTheme = BlackTheme
    game = GameAction(self.screen, gameTheme, self.loop, boardSize)
    game.runGameAI(aiLevel)
```

Thực hiện tạo và gọi menu trong hàm chính chương trình:

```
import pygame
from GameSettings.DefaultSettings import *
from Menu.MainMenu import Menu

pygame.init()
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('Trò chơi cờ caro')

if __name__ == "__main__":
    menu = Menu(screen)
    menu.run()
```

Link github: https://github.com/huykhaduy/PythonCaroGame

Một số hình ảnh Demo:



TÀI LIỆU THAM KHẢO

[1] Coding Spot, *"Coding an Unbeatable Tic Tac Toe AI Using Python and the Minimax Algorithm"*, https://www.youtube.com/watch?v=Bk9hlNZc6sE&ab_channel=CodingSpot

[2] Sebastian Lague, *"Algorithms Explained – minimax and alpha-beta pruning"*, https://www.youtube.com/watch?v=l-hh51ncgDI&ab_channel=SebastianLague

[3] Pygame_menu, *"Pygame menu documents"*, https://pygame-menu.readthedocs.io/en/4.2.8/

[4] Pygame, "Pygame documents", https://www.pygame.org/docs/

[5] Coding Spot, *"Python tictactoe ai"*, https://github.com/AlejoG10/python-tictactoe-ai-yt