Báo cáo về bài thực hành Lab 3

Tìm kiếm đối kháng

Họ và tên người thực hiện: Lưu Ngọc Khang
MSSV: 22880068

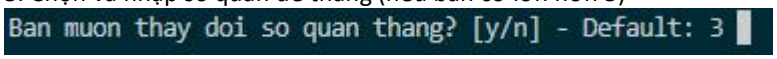
Công việc đã thực hiện:

- Đọc hiểu, diễn giải code trên github và source code minimax.py
- Cài đặt source code minimax.py của github và chạy cho trường hợp cơ bản (3x3)
- Thay đổi UI cho phù hợp với yêu cầu đề bài (mở rộng bàn cờ)
- Giảm số nước đi nhìn trước để tăng thời gian phản hồi từ AI (có thể làm AI chơi kém hiệu quả)
- Áp dụng tỉa nhánh alpha-beta vào minimax
(Tham khảo https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/)
- Cài đặt chế độ set trước bàn cờ (có thể dùng để chơi 2 người, chưa hỗ trợ đọc từ file text bàn cờ)
- Cài đặt mở rộng bàn cờ và set số quân thắng

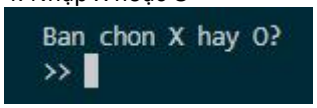
Hướng dẫn chạy thử

1. Chạy code
2. Nhập kích thước bàn cờ

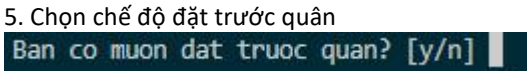
3. Chọn và nhập số quân để thắng (nếu bàn cờ lớn hơn 3)



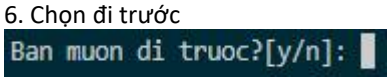
4. Nhập X hoặc O



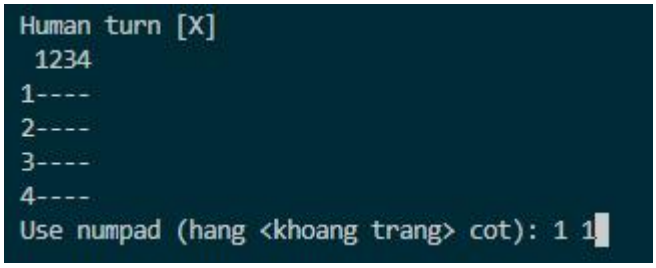
5. Chọn chế độ đặt trước quân



6. Chọn đi trước



7. Chọn ô để đi theo cú pháp tọa độ [x, y]



Have fun!

Chi tiết

Phần 1 - Đọc hiểu code thuật toán Minimax trên nguồn có sẵn

Nguồn : https://github.com/Cledersonbc/tic-tac-toe-minimax

I. Pseudo code:

```
minimax(state, depth, player)

        if (player = max) then
                best = [null, -infinity]
        else
                best = [null, +infinity]

        if (depth = 0 or gameover) then
                score = evaluate this state for player
                return [null, score]

        for each valid move m for player in state s do
                execute move m on s
                [move, score] = minimax(s, depth - 1, -player)
                undo move m on s

                if (player = max) then
                        if score > best.score then best = [move, score]
                else
                        if score < best.score then best = [move, score]

        return best
end
```

II. Diễn giải mã nguồn dựa trên trình bày của github:

1. Khai báo các biến bàn cờ, max và min.

```
board = [ [0, 0, 0], [0, 0, 0], [0, 0, 0] ]
MAX = +1
MIN = -1
```

2. Khai báo hàm minimax:

```
def minimax(state, depth, player):
```

Hàm minimax nhận 3 thông số:

- state: trạng thái bàn cờ hiện tại (node)
- depth: chỉ số độ sâu (chiều cao) của node
- player: người chơi là max hay min

3. Khai báo và khởi tạo cho biến best:

```
if player == MAX:
        best = [-1, -1, -infinity]
else:
        best = [-1, -1, +infinity]
```

- best là list có 3 giá trị: index 0, 1 lần lượt lưu tọa độ x (hàng), y (cột) trên bàn cờ, và index 2 lưu giá trị điểm.
- Nếu node hiện tại là lượt đi của người chơi max, khởi tạo các nước đi tốt nhất và điểm tệ nhất lần lượt là (-1, -1) và -vô cực
- Ngược lại nếu là người chơi min, nước đi tốt nhất là (-1, -1) và điểm là +vô cực.
- Khởi tạo nước đi là (-1, -1) là vì bàn cờ là một ma trận 3x3, có chỉ số bắt đầu từ 0. Nên (-1, -1) là nước đi không hợp lệ, tương đương với null trong pseudocode.

Đối chiếu với lý thuyết:

- -vô cực là được khởi tạo là điểm tệ nhất cho người chơi max, còn +vô cực là điểm tệ nhất cho người chơi min.
- Lý do là người chơi min sẽ so sánh và chọn giá trị nhỏ nhất từ trạng thái con, còn người chơi max sẽ chọn giá trị lớn nhất. Khởi tạo như vậy mới giúp người chơi tương ứng so sánh và cập nhật được điểm số trong code.
- ~~Hơn nữa, các giá trị này tương đương với cận dưới (alpha) của max và cận trên (beta) của min khi đối chiếu với lý thuyết tỉa nhánh alpha-beta.~~

4. Khi đạt depth = 0 hoặc đến được trạng thái trò chơi kết thúc:

```
if depth == 0 or game_over(state):
        score = evaluate(state)
        return score
```

Depth là 0 thì trên bàn cờ không còn ô trống, hoặc khi một người chơi đã thắng thì trò chơi đã kết thúc khi đến lượt của MAX hoặc MIN. Lúc đó, trả giá trị điểm như sau trong hàm lượng giá evaluate tại trạng thái (state) hiện tại:
Nếu MAX thắng: trả +1.
Nếu MIN thắng: trả -1.
Ngược lại: trả 0 (hòa)

Đối chiếu với lý thuyết:

- Theo lý thuyết tìm kiếm đối kháng zero-sum, người chơi MAX tối đa hóa giá trị của mình, và người chơi MIN tối thiểu hóa giá trị của đối thủ. Người này được thì người kia mất.

5. Duyệt các nước đi hợp lệ tiếp theo hay còn gọi là các trạng thái con của trạng thái hiện tại và đệ quy:

```
for cell in empty_cells(state):
        x, y = cell[0], cell[1]
        state[x][y] = player
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y
```

Lặp lại đối với các nước đi hợp lệ là ô trống được xác định bằng hàm empty_cells với trạng thái (state) hiện tại:
- Biến cell một ô trong các ô trống trong node hiện tại.
- x, y là tọa độ hàng, cột của cell trống đó
- state[x][y] = player: Trong biến state lưu trạng thái bàn cờ, gán giá trị cho vị trí cell tại (x, y) là player, nghĩa là ô này được người chơi hiện tại chọn cho trạng thái này.
- score = minimax(state, depth - 1, -player): Đệ quy hàm minimax cho state trạng thái tiếp theo và trả giá trị vào biến score.
    với state: có ô tại (x, y) được đánh dấu người chơi của trạng thái trước,
    và depth - 1: là chỉ số của trạng thái tiếp theo
    và -player: để đảo người chơi MAX (+1) thành MIN(-1) và ngược lại.

Tóm lại là các dòng trên thực hiện việc ghi nhận giá trị điểm của nước đi tiếp theo.

- Sau khi duyệt hết tất các trạng thái con, gán 0 cho state[x][y], coi như “lui về” từ bước đó.
- Gán x, y cho score[0], score[1] để lưu lại tại cell này sẽ ăn được điểm gì.
(score là [hàng, cột, điểm])

Đối chiếu với lý thuyết:

- Việc đệ quy hàm minimax này là duyệt theo chiều sâu (DFS), giống như lúc thực hiện tìm kiếm đối kháng theo minimax bằng tay ta cũng tính các giá trị tại node min/max từ dưới lên.
- Sau khi tìm được giá trị của trạng thái (node) min/max bằng hàm minimax thì ta ghi nhận giá trị tại node đó. Trong code thì lưu vào biến score rồi

6. So sánh score và best sau khi tìm được score tại cell đang xét:

```
if player == MAX:
        if score[2] > best[2]:
                best = score
else:
        if score[2] < best[2]:
                best = score
```

Tìm xong score tại cell đó thì cập nhật best:

Nếu player ở trạng thái hiện tại là MAX, và điểm tại ô đó (score) cao hơn điểm cao nhất (best) thì cập nhật best là score

Và làm tương tự với trường hợp ngược lại nếu player là MIN và score bé hơn best.

III. Diễn giải code dựa trên file source code minimax.py trong link Source:

Source code có định nghĩa các hàm như evaluate, wins,… sau đây:

1. Hàm evaluate - Hàm lượng giá:

```
def evaluate(state):
    """
    Function to heuristic evaluation of state.
    :param state: the state of the current board
    :return: +1 if the computer wins; -1 if the human wins; 0 draw
    """
    if wins(state, COMP):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else:
        score = 0

    return score
```

Gọi hàm wins, truyền vào người chơi và trạng thái (bàn cờ) hiện tại để xác định trạng thái hiện tại ai đang thắng, hay đang hòa. Lưu ý trong source code này thay thế MAX là COMP (máy) còn MIN là HUMAN (người).

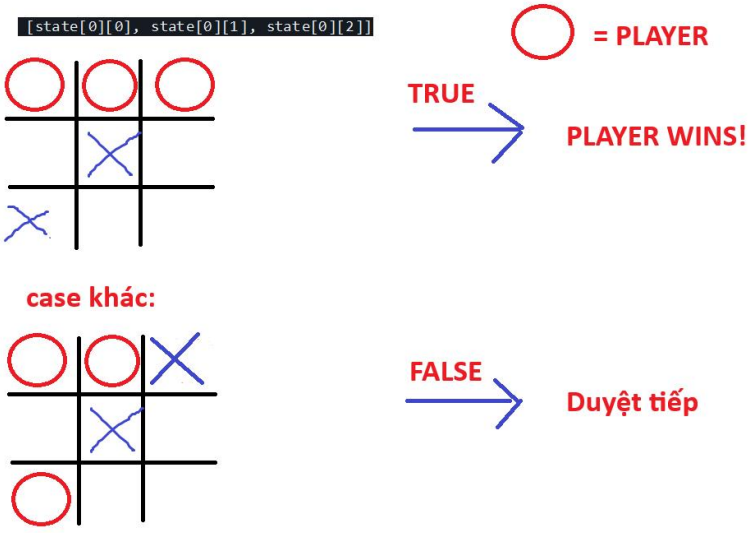
2. Hàm wins - Xác định trạng thái thắng của trò chơi:

```
def wins(state, player):
    """
    This function tests if a specific player wins. Possibilities:
    * Three rows    [X X X] or [O O O]
    * Three cols    [X X X] or [O O O]
    * Two diagonals [X X X] or [O O O]
    :param state: the state of the current board
    :param player: a human or a computer
    :return: True if the player wins
    """
    win_state = [
        [state[0][0], state[0][1], state[0][2]],
        [state[1][0], state[1][1], state[1][2]],
        [state[2][0], state[2][1], state[2][2]],
        [state[0][0], state[1][0], state[2][0]],
        [state[0][1], state[1][1], state[2][1]],
        [state[0][2], state[1][2], state[2][2]],
        [state[0][0], state[1][1], state[2][2]],
        [state[2][0], state[1][1], state[0][2]],
    ]
    if [player, player, player] in win_state:
        return True
    else:
        return False
```

Hàm wins có biến danh sách win_state để chỉ ra các trạng thái thắng diễn giải như sau:
3 dòng đầu mô tả “3 quân nằm trên một đường thẳng”
3 dòng sau mô tả “3 quân nằm trên một hàng dọc”
2 dòng sau mô tả “3 quân nằm trên một đường chéo”

Nếu một trong bất kì dòng trên có kết quả là [player, player, player], thì người chơi thắng và trả về True. Nếu không có kết quả trùng thì trả về False.

Ví dụ minh họa:



3. Hàm game_over - Hàm kiểm tra người hay máy đã thắng chưa:

```
def game_over(state):
    """
    This function test if the human or computer wins
    :param state: the state of the current board
    :return: True if the human or computer wins
    """
    return wins(state, HUMAN) or wins(state, COMP)
```

Gọi hàm wins.

4. Hàm empty_cells - Trả về một danh sách các ô còn trống

```
def empty_cells(state):
    """
    Each empty cell will be added into cells' list
    :param state: the state of the current board
    :return: a list of empty cells
    """
    cells = []

    for x, row in enumerate(state):
        for y, cell in enumerate(row):
            if cell == 0:
                cells.append([x, y])

    return cells
```

Nhận trạng thái bàn cờ được truyền vào.

- Khởi tạo list cells để index các ô còn trống.
- Hàm enumerate là hàm đánh stt là chỉ số các phần tử trong list của python.
- for x, row in enumerate:
    - x lưu chỉ số của các dòng trên bàn cờ,
    - row là dòng tại x (mỗi dòng là một list 3 phần tử, lưu giá trị của player (là 0 nếu chưa điền, +1 nếu COMP, -1 nếu HUMAN).
    -> Lấy các dòng có đánh số thứ tự (hoạt động như chỉ số tọa độ dòng)
- for y, cell in enumerate (row):
    - y lưu chỉ số của các cột trong row.
    - cell là giá trị của từng phần tử trong row, hay còn gọi là một ô.
    -> Lấy các ô trong dòng có đánh số thứ tự (chỉ số tọa độ cột)
- if cell == 0 :
    cells.append([x,y])

-> Nếu cell này có giá trị là 0 nghĩa là chưa ai điền thì thêm list [x, y] vào list cells.

5. Hàm valid_move - Hàm kiểm tra nước đi hợp lệ

```
    def valid_move(x, y):
        """
        A move is valid if the chosen cell is empty
        :param x: X coordinate
        :param y: Y coordinate
        :return: True if the board[x][y] is empty
        """
        if [x, y] in empty_cells(board):
            return True
        else:
            return False
```

Nhận x, y là tọa độ dòng, cột.

- if [x, y] in empty_cells(board): Nếu tại [x, y] là một ô trống thì hợp lệ, trả True. Ngược lại trả False.

6. Hàm set_move - Hàm đặt quân lên bàn cờ

```
    def set_move(x, y, player):
        """
        Set the move on board, if the coordinates are valid
        :param x: X coordinate
        :param y: Y coordinate
        :param player: the current player
        """
        if valid_move(x, y):
            board[x][y] = player
            return True
        else:
            return False
```

Gọi hàm valid_move để kiểm tra tại (x, y) có phải là nước đi hợp lệ hay không.

7. Hàm minimax (giống Diễn giải mã nguồn dựa trên trình bày của github:)
8. Các hàm đồ họa:

clean - xóa toàn bộ text

render - in bàn cờ

9. Hàm ai_turn - xử lý lượt AI:

```
        depth = len(empty_cells(board))
        if depth == 0 or game_over(board):
            return
```

Gọi empty_cell để kiểm tra còn bao nhiêu ô, và gọi game_over để kiểm tra trò chơi đã kết thúc chưa.

```
        clean()
        print(f'Computer turn [{c_choice}]')
        render(board, c_choice, h_choice)
```

Nếu chưa thì xóa màn hình và in thông báo + bàn cờ ra màn hình.

```
        if depth == 9:
            x = choice([0, 1, 2])
            y = choice([0, 1, 2])
        else:
            move = minimax(board, depth, COMP)
            x, y = move[0], move[1]

        set_move(x, y, COMP)
        time.sleep(1)
```

Sau đó thực hiện chọn ngẫu nhiên x và y nếu chưa có nước đi nào trên bàn, nếu đã có thì gọi hàm minimax để xử lý. Sau đó gọi set_move để đi.

10. Hàm human_turn : xử lý nước đi của người chơi (con người)

```
        depth = len(empty_cells(board))
        if depth == 0 or game_over(board):
            return
```

Kiểm tra trạng thái bàn cờ.

```
        # Dictionary of valid moves
        move = -1
        moves = {
            1: [0, 0], 2: [0, 1], 3: [0, 2],
            4: [1, 0], 5: [1, 1], 6: [1, 2],
            7: [2, 0], 8: [2, 1], 9: [2, 2],
        }
```

Tạo biến move = -1, dictionary moves cho các lựa chọn hợp lệ.

```
        clean()
        print(f'Human turn [{h_choice}]')
        render(board, c_choice, h_choice)
```

Xóa màn hình, thông báo, vẽ bàn cờ.

```
        while move < 1 or move > 9:
            try:
                move = int(input('Use numpad (1..9): '))
                coord = moves[move]
                can_move = set_move(coord[0], coord[1], HUMAN)

                if not can_move:
                    print('Bad move')
                    move = -1
            except (EOFError, KeyboardInterrupt):
                print('Bye')
                exit()
            except (KeyError, ValueError):
                print('Bad choice')
```

Xử lý nhập liệu, cho người chơi chọn bằng cách nhập số từ 1 tới 9 và kiểm tra bằng hàm set_move. Thoát chương trình nếu dữ liệu nhập không hợp lệ (input là string) hay nhấn tổ hợp Ctrl + C.

Phần 2 - Cài đặt code

1. Thiết kế bàn cờ

- Để cho việc điều chỉnh kích thước bàn cờ sau này, em tối giản hóa UI bằng cách tạo thêm hàm create_board như sau để phát sinh tự động bàn cờ theo kích thước N x N

```
    def create_board(board_size):
        board = [[0] * board_size]*board_size
```

Tuy nhiên, cú pháp trên sẽ tạo ra một list gồm các list có ID giống nhau, dẫn tới bug khi update một phần tử trong list con sẽ update toàn bộ các list khác như sau:

```
Human turn [X]
 12345
1--X--
2--X--
3--X--
4--X--
5--X--
YOU WIN!
```

Do đó cú pháp được viết lại thành như dưới đây, theo tham khảo trên stackoverflow về ID list:
https://stackoverflow.com/questions/240178/list-of-lists-changes-reflected-across-sublists-unexpectedly

```
        board = [[0] * board_size for _ in range(board_size)]
        # This creates a list with many lists of different IDs
```

Kết quả:

```
Computer turn [O]
 12345
1-----
2-----
3-----
4-----
5----X
```

- Bàn cờ gồm N x N ô, với các cột được đánh số thứ tự từ 1 đến N bằng hàm render_column_indexes được tạo thêm (trong source)
Tiếp theo cần số thứ tự hàng kèm với nội dung hàng bằng hàm render được sửa lại từ hàm gốc (

2. Xử lý lượt chơi cho người:

- Người dùng nhập input theo format trong lượt của mình: [số hàng, số cột]
- Số nhập vào được trừ 1 khi xử lý dưới code. Sử dụng lại hàm set_move, valid_move có sẵn trong nguồn mẫu để biết nước đi hợp lệ, vì nó truy xuất list bằng giá trị hàng/cột thay vì đánh số ô

3. Điều kiện chiến thắng:

- Có 4 trường hợp thắng: Có N ô liên tiếp của người chơi trên một trong các đường thẳng sau:
    - Đường ngang: x = x, y = y + i
    - Đường dọc: x = x + i, y = y
    - Đường chéo lên / : x = x - i, y = y + i
    - Đường chéo xuống \ : x = x + i, y = y - i

- Cách xét: Lấy trong state tập hợp ô của người chơi, và xét từng ô trên bàn cờ, nếu có quân nằm trên đường thẳng, dọc, chéo phải, chéo trái thì tăng biến đếm cho đường thẳng đó.
(Có tham khảo code từ nguồn khác: Hàm victory_check từ https://github.com/dwaltrip/m-n-k-game/blob/master/tictactoe.py)

- Hàm lấy tọa độ ô người chơi

```
    def get_player_cell_coords(state,player):
        player_cell_coords = []
        for i in range(0, len(state)):          # index of row
            for j in range(0,len(state[i])):    # index of column
                if state[i][j] == player:
                    player_cell_coords.append([i,j])  # coordination of
cell
        return player_cell_coords
```

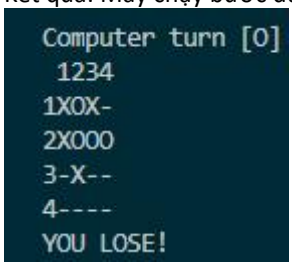
- Hàm xét đường thẳng:

4. Cải thiện tốc độ AI:

- Thêm biến initial_depth để so sánh với depth hiện tại, giới hạn số bước máy nhìn trước.
VD: Chỉ nhìn trước 5 bước

```
        if depth == 0 or depth < initial_depth-5 or game_over(state):
            score = evaluate(state)
            return [-1, -1, score]
```

Kết quả: Máy chạy bước đầu sau người phải mất khoảng 2-3 phút cho bàn cờ 4x4, số quân thắng là 3.



- Tỉa nhánh Alpha-beta:

Thêm biến alpha beta và xét nếu người chơi là máy (người chơi MAX) thì gán alpha là best (giá trị lớn nhất), nếu beta bé hơn hoặc bằng alpha thì không xét tiếp trạng thái con. Ngược lại nếu người chơi là người thì gán beta là best (giá trị nhỏ nhất), nếu beta <= alpha thì break.

Sau khi áp dụng tỉa nhánh kết hợp với giới hạn số bước nhìn trước, tốc độ cải thiện từ 3 phút xuống khoảng 3-5 giây cho bước thứ hai sau người.

```
        for cell in empty_cells(state):
            x, y = cell[0], cell[1]
            state[x][y] = player
            score = minimax(state, depth - 1, -player, initial_depth, alpha
beta)
            state[x][y] = 0
            score[0], score[1] = x, y

            if player == COMP:
                if score[2] > best[2]:
                    best = score  # max value
                if best[2] > alpha[2]:
                    alpha = best
                if beta[2] <= alpha[2]:
```

```
                    break
            else:
                if score[2] < best[2]:
                    best = score  # min value
                if best[2] < beta[2]:
                    beta = best
                if beta[2] <= alpha[2]:
                    break
```

5. Cài đặt chiều rộng bàn cờ:

Máy yêu cầu người dùng nhập chiều dài bàn cờ và tạo bàn cờ hình vuông N x N.

```
    def input_board_size():
        board_size = -1
        while board_size < 3 or board_size > 100:
                try:
                    print('')
                    board_size = int(input('Nhap kich thuoc ban co [3..100]
'))

                    if (board_size < 3 or board_size > 100):
                        print('Vui long nhap tu 3 den 100')

                except (EOFError, KeyboardInterrupt):
                    print('Bye')
                    exit()
                except (KeyError, ValueError):
                    print('Input khong hop le')
        board[:] = create_board(board_size)

    def create_board(board_size):
        board = [[0] * board_size for _ in range(board_size)]
        # This creates a list with many lists of different IDs
        print(board)
        return board
```

6. Cài đặt số quân thắng:

Số quân thắng mặc định là 3. Nếu bàn cờ có kích thước lớn hơn 3, máy sẽ hỏi người dùng có muốn set số quân thắng hay không.

```
    def set_num_to_win():

        selection = ''
        while selection != 'Y' and selection != 'N':
            try:
                print('')
                selection = input('Ban muon thay doi so quan thang? [y/n] -
Default: {} '.format(number_to_win)).upper()

            except (EOFError, KeyboardInterrupt):
                print('Bye')
                exit()
            except (KeyError, ValueError):
                 print('Input khong hop le')
        num = 0

        while num < 3 or num > len(board):
```

```
                try:
                    print('')
                    num = int(input('Nhap so quan thang [3...{}]:
'.format(len(board))))

                    if (num < 3 or num > len(board)):
                        print('Vui long nhap tu 3 den
{}'.format(len(board)))

                except (EOFError, KeyboardInterrupt):
                    print('Bye')
                    exit()
                except (KeyError, ValueError):
                    print('Input khong hop le')

        return num
```

Điểm có thể cải thiện

- Người chơi có thể chọn theo ô được highlight thay vì nhập tọa độ.
- Hàm lượng giá cho máy dựa trên hàng không bị chặn để đánh giá được điểm của trạng thái trước trạng thái kết thúc để tăng độ tối ưu cho nước đi của máy. Dùng chung với hàm giới hạn tầm nhìn nước đi của máy.
- Trong một source code khác, người ta còn áp dụng thuật toán Monte Carlo Tree Search, nhưng do chưa hiểu nó là gì nên trong source này chưa cài đặt. https://github.com/ae-bii/mnk-game-ai