MIN MAX SEARCH
ALPHA BETA ALGORITHM
CARO - GAME

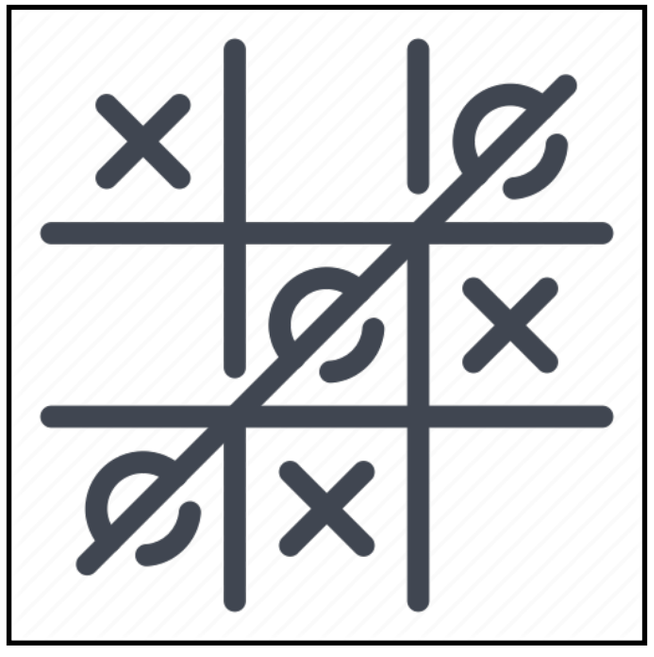
Người thực hiện: Trần Ngọc Bảo Duy - 51702091
Phạm Anh Duy - 51702088

Người hướng dẫn: PGS TS Lê Anh Cường

1. MinMaxSearch Algorithm

1. Khái niệm

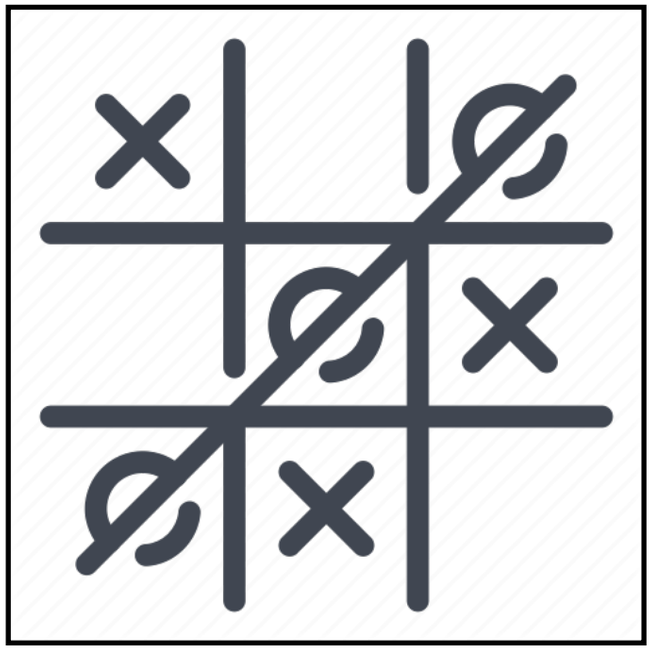
- Minimax search là một thuật toán nhằm tối thiểu hoá các tổn thất (giá trị mất mát), hoặc có thể hiểu là tối đa hoá lợi ích (giá trị tốt) trong các nước đi được tính toán trước
- Giải thuật Minimax giúp tìm ra nước đi tốt nhất, bằng cách đi ngược từ cuối trò chơi trở về đầu. Tại mỗi bước, nó sẽ ước 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).
- Ví dụ như trò tic tac toe, với mỗi bước đi player sẽ tính toán các bước đi để tối đa cơ hội thắng của mình và tối thiểu cơ hội thắng cuả đối phương



1. MinMaxSearch Algorithm

2. Áp dụng

- Ta sẽ cần một hàm evaluate (thường là hàm đệ quy) để đánh giá số điểm của 1 nước đi cho trước bới 1 trạng thái nhất định của người chơi
- Sau khi evaluate, ta sẽ chọn nước đi có cơ hội thắng và cơ hội để đối thủ thua cao nhất. Hàm này sẽ tự tạo ra các trạng thái kế tiếp của cả người chơi và đối thủ để tính toán nước đi mà đối thủ có thể chọn.
- Nếu các bước evaluate tiến dần về vô hạn đối với mỗi nước đi kế tiếp, thì người chơi có thể chọn ngẫu nhiên các nước đi tiên phong, tránh trường hợp vô cực.
- Ví dụ như trò tic tac toe, nếu không có nước đi trên bàn cờ người chơi có thể chọn ngẫu nhiên trước khi thuật toán evaluate được kích hoạt



1. MinMaxSearch Algorithm

3. Mã giả

```
MinMaxSearch(player)
        { // player là nút muốn tính điểm
        If  depth is 0: (độ sâu mà chúng ta muốn tính)
                return score of  player
        else:
                If  player in a MIN_PLAYER
                        For all_next_move of  player: v1,…,vn
                        Return   min { MinMaxSearch (v1),…, MinMaxSearch (vn)}
                Else (player is MAX_PLAYER)
                        For all_next_move of player: v1,…,vn
                        Return   max { MinMaxSearch (v1),…, MinMaxSearch (vn)}
}
```

2. Alpha beta Algorithm

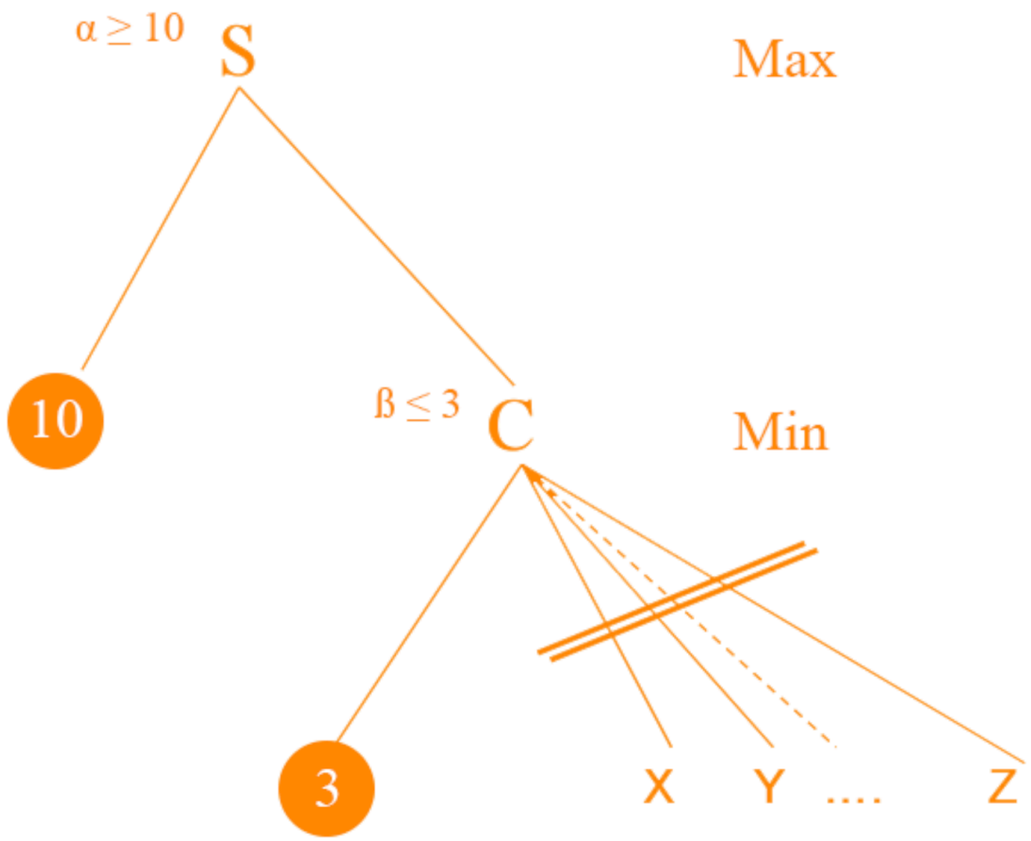
1. Khái niệm

- Alpha beta Algorithm là một thuật toán nâng cấp của Minmaxsearch thay vì chúng ta phải tìm tất cả các nước đi trên một bàn cờ cho đến độ sâu mà chúng ta muốn, điều này sẽ dẫn đến overload RAM vì không thể nào đệ quy hết được tất cả các nước đi, ở độ sâu 2 các nước đi đã dần chậm chạp so với các nước ban đầu, nếu tính lên độ sâu 5,6 thậm chí là 10 máy tính chúng ta sẽ không chạy được hoặc thời gian chờ đợi sẽ rất lâu, vì vậy thuật toán alpha beta đã xuất hiện để giải quyết vấn đề này
- Chúng ta đều biết rằng khi thuật toán minmax nó sẽ tìm ra được các ngưỡng trên và ngưỡng dưới của nước đi tuỳ theo người chơi là min hay max.
- Chính điều này có thể giúp chúng ta chặn trước để có thể làm giảm các tính toán của máy tính, ta sẽ dùng 2 biến ngưỡng trên và ngưỡng dưới [alpha, beta] tại mỗi node để tính toán và kiểm tra có nên đi evaluate tiếp hướng đi (tree structure) đó không hay quay lại.

2. Alpha beta Algorithm

2. Áp dụng

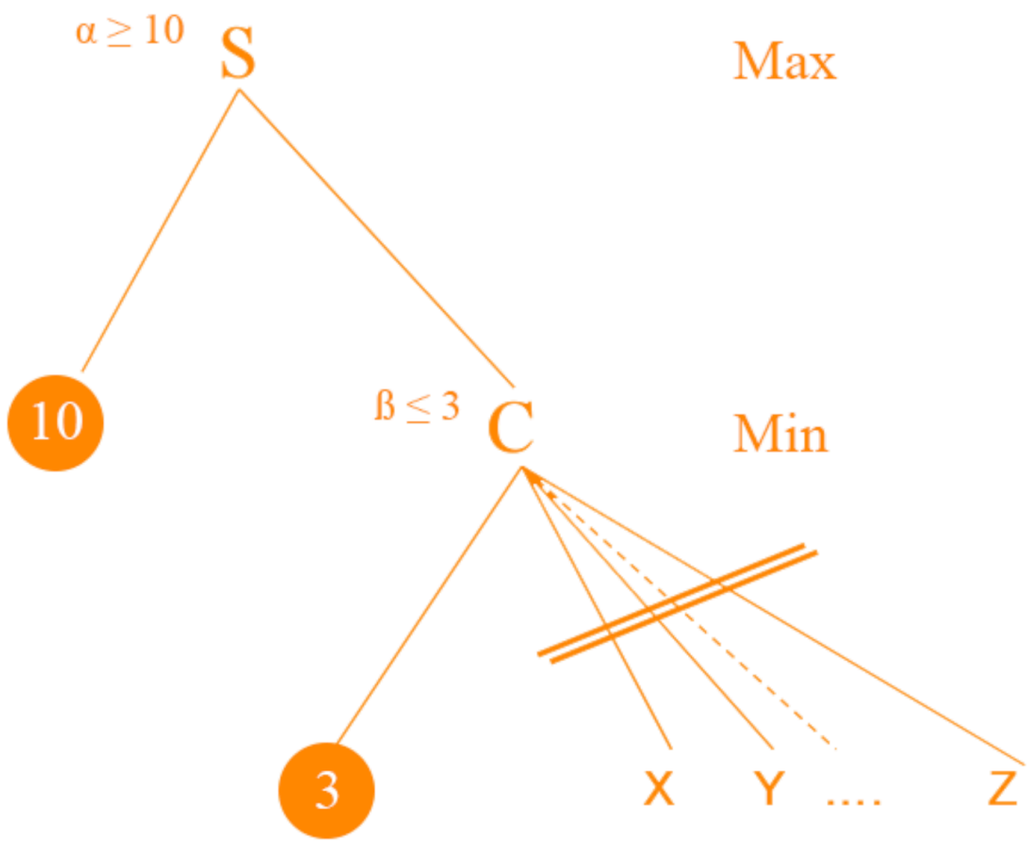
- Ta thấy người chơi cần đánh giá là Max cho nên các node kế tiếp phải lấy giá trị max
- Đi xuống node đầu tiên giá trị là 10, vì người chơi mà Max nên ta đặt lại khoảng [alpha, beta] là [10, beta]
- Tiếp tục đi node kế tiếp, vì node kế tiếp có con nên chúng ta phải tính tiếp, xuống tới node 3 vì đây là người chơi min nên chúng ta sẽ đặt lại khoảng [alpha, beta] là [alpha, 3]



2. Alpha beta Algorithm

2. Áp dụng

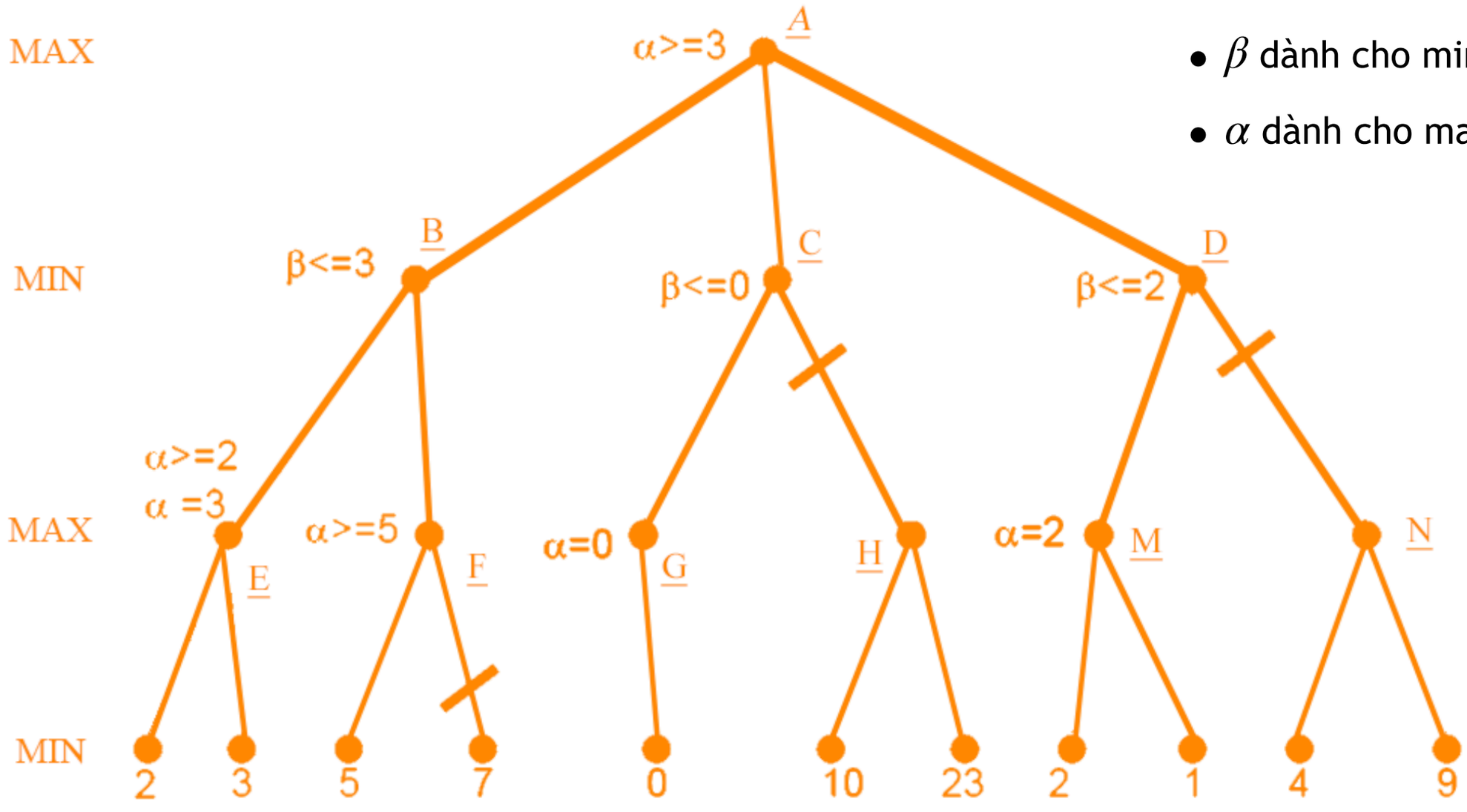
- Sau khi xác định được alpha và beta, chúng ta có thể dễ dàng xác định việc có cắt tỉa hay không. Ở nút S (Max), giá trị alpha luôn ≥ 10 (luôn tăng) nhưng ở C (Min) thì giá trị luôn luôn ≤ 3 (luôn giảm), nên việc xét các con còn lại ở C là không cần thiết. Vì ở S chúng ta sẽ nhận giá trị Max luôn phải lớn hơn bằng 10 cho nên giá trị nhỏ hơn bằng 3 kia không có ý nghĩa nữa



2. Alpha beta Algorithm

2. Áp dụng

- Ví dụ khác về cây sử dụng alpha beta để cắt nhánh, tránh tính trạng quá tải trong các nút



- β dành cho min
- α dành cho max

2. Alpha beta Algorithm

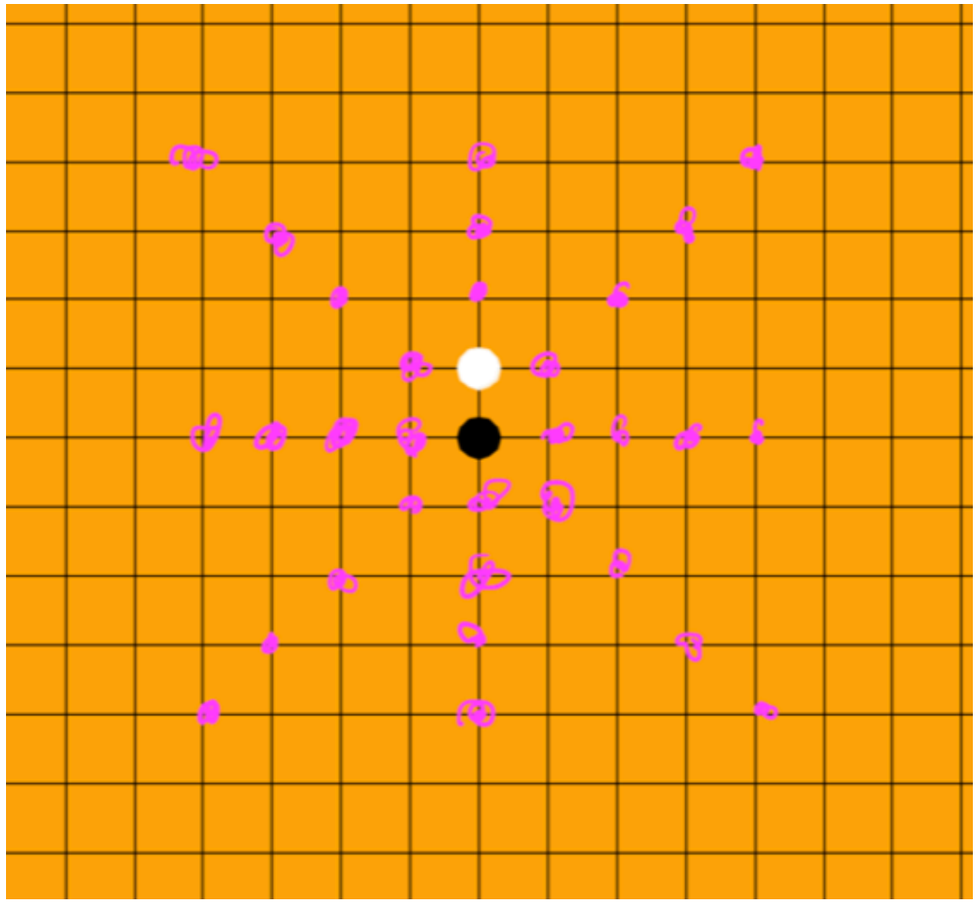
3. Mã giả

```
if depth = 0 then
        AlphaBeta = Eval  // Tính điểm tại vị trí node lá
else
        {
           best = -INFINITY; // cho best là âm vô cực để chặn các giá trị trên
           Generate;  //Sinh ra các node con có thể đi
                  while (còn lấy được node để đi) and (best < beta) do
                  {
                        if best > alpha then:
                              alpha = best;
                              đi node con kế tiếp để tính value dự đoán
                              value = -AlphaBeta(-beta, -alpha, depth-1);
                              lấy lại node ban đầu, lúc chưa đi node kế tiếp để tính value ngay tại
                              node đó
                       if value > best then:
                              best = value;
                  }
          AlphaBeta = best;
        }
}
```

3. Hiện thực code

Những method cần quan tâm:

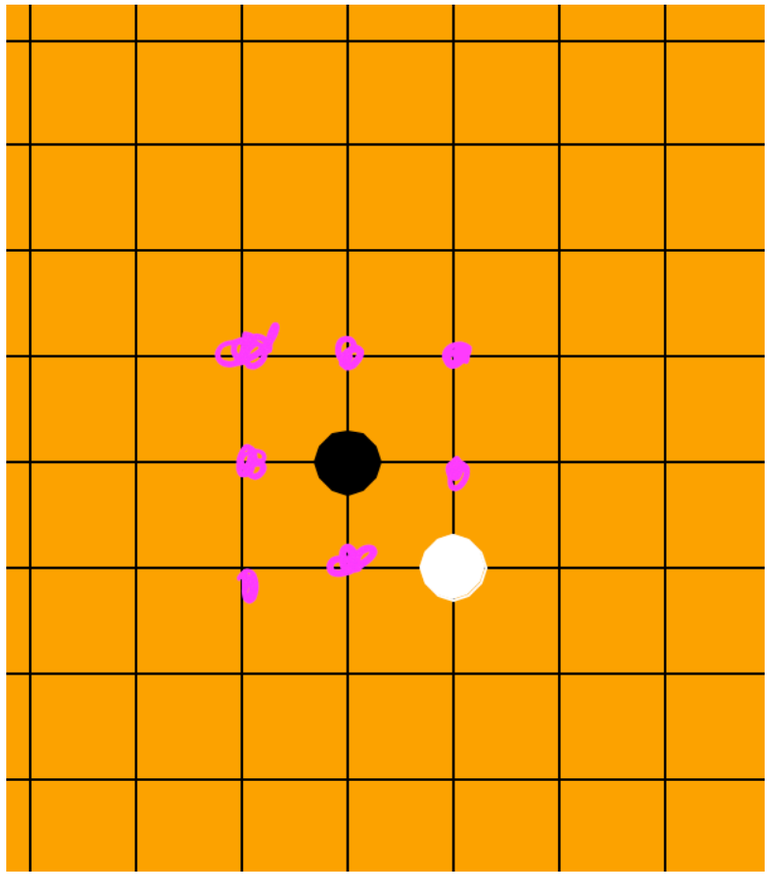
- **Phạm Duy:**
  - get_all_move()
  - is_win()
  - find_index_near_to_player()
- **Trần Duy:**
  - get_possible_move_to_cal_score()
  - call_score()
  - play()
- **Cả hai:**
  - **minmaxSearch()**
  - **alpha_beta()**
  - design lại giao diện



Lấy tất cả những hướng đi có thể của người chơi

```
# lấy tất cả 8 hướng đi
def get_all_move(poisition,board):
    x, y = poisition
    player = board[x, y]
    eight_dir= {'up':[],'up_right':[],'right':[],'down_right':[],'down':[],'down_left':[],'left':[],'up_left':[]}
    for i in range(1,5):
        max_step = [[x + y*i for x, y in zip(poisition, ele)] for ele in eight_direct]
        for index,key in enumerate(eight_dir.keys()):
            next_x, next_y = max_step[index]
            if is_in_board(max_step[index]) != False:
                eight_dir[key].append(max_step[index])
    return eight_dir
```

3. Hiện thực code



Lấy tất cả những hướng nước đi xung quanh để tính điểm

```python
# lấy tất cả 8 hướng đi
def get_possible_move_to_cal_score(poisition,board,is_min_player):
    x, y = poisition
    eight_dir= {'up':[],'up_right':[],'right':[],'down_right':[],'down':[],'down_left':[],'left':[],'up_left':[]}
    for i in range(1,2):
        max_step = [[x + y*i for x, y in zip(poisition, ele)] for ele in eight_direct]
        for index,key in enumerate(eight_dir.keys()):
            next_x, next_y = max_step[index]
            if is_in_board(max_step[index]) != False and board[next_x,next_y] ==0:
                eight_dir[key].append(max_step[index])
    return eight_dir
```

3. Hiện thực code

- Hàm kiểm tra hướng đã win chưa nếu win trả về True, ngược lại False
- Hàm này sẽ tính điểm theo 8 hướng, hướng nào có điểm cao nhất là >= 5 thì return True

```python
# sau đó tỉnh tổng của trên dưới và 2 đường chéo, nếu lớn hơn 5 là win
def is_win(_board,poisition,isMinPlayer):
    board = np.copy(_board)
    x, y = poisition
    player = board[x, y]
    if isMinPlayer:
        board[x, y] = 1
    else:
        board[x, y] = 2
    if isMinPlayer:
        player = PLAYER_MIN
        opponent = PLAYER_MAX
    else:
        player = PLAYER_MAX
        opponent = PLAYER_MIN
    score = {'horizontal':1,'vertical':1,'diagonal_1':1,'diagonal_2':1}
    all_move = get_all_move(poisition,board)
    # check liên tục
    continue_move = [True,True,True,True,True,True,True,True]
    for key, moves in all_move.items():
        for next_move in moves:
            if is_in_board(next_move):
                x, y = next_move
                _score = get_score(next_move, player)
                # kiểm tra trên 4 đường nếu tổng đủ hơn 5, nếu bị chặn bởi opponent
                # hoặc ra khỏi board sẽ ngừng đếm

                if key == 'up_left' and  continue_move[0]:
                    if board[x,y] == opponent or board[x,y] != player: continue_move[0] = False
                    else: score['diagonal_1'] += _score
                elif key == 'up' and continue_move[1]:
                    if board[x,y] == opponent or board[x,y] != player : continue_move[1] = False
                    else: score['vertical'] += _score
                elif key == 'up_right'and continue_move[2]:
                    if board[x,y] == opponent or board[x,y] != player:continue_move[2] = False
                    else: score['diagonal_2'] += _score
                elif key == 'right' and continue_move[3]:
                    if board[x,y] == opponent or board[x,y] != player: continue_move[3] = False
                    else: score['horizontal'] += _score
                elif key == 'down_right' and continue_move[4]:
                    if board[x,y] == opponent or board[x,y] != player: continue_move[4] = False
                    else: score['diagonal_1'] += _score
                elif key == 'down' and continue_move[5]:
                    if board[x,y] == opponent or board[x,y] != player: continue_move[5] = False
                    else: score['vertical'] += _score
                elif key == 'down_left' and continue_move[6]:
                    if board[x,y] == opponent or board[x,y] != player: continue_move[6] = False
                    else: score['diagonal_2'] += _score
                elif key == 'left' and continue_move[7]:
                    if board[x,y] == opponent or board[x,y] != player: continue_move[7] = False
                    else: score['horizontal'] += _score
    for item in score.values():
        if item >= 5:
            return True
    return False
```

```python
def v_score(scores):
    dic = {0:0,1:0,2:0,3:0,4:0,5:0}
    for key, item in scores.items():
        if item in dic.keys():
            dic[item] +=1
    return dic
```

- Hàm v_score() thống kê lại điểm của các hướng như là:

  {

  0: bao nhiêu hướng,

  1: bao nhiêu hướng,

  … ,

  5: bao nhiêu hướng

  }

- Hàm call_score() hàm tính điểm, giải thích ở hình slide kế tiếp

```python
def call_score(_board,poisition,is_min_player):
    board = np.copy(_board)
    i, j = poisition
    player = board[i, j]
    if is_min_player:
        player = PLAYER_MIN
        opponent = PLAYER_MAX
    else:
        player = PLAYER_MAX
        opponent = PLAYER_MIN
    # lưu điểm của mỗi hướng
    score_every_direc = {'up':1,'up_right':1,'right':1,'down_right':1,'down':1,'down_left':1,'left':1,'up_left':1}
    all_move = get_all_move(poisition,board)
    # check liên tục
    continue_move = [True,True,True,True,True,True,True,True]
    bi_chan = {'horizontal':False,'vertical':False,'diagonal_1':False,'diagonal_2':False}
    for key, moves in all_move.items():
        for next_move in moves:
            if is_in_board(next_move):
                x, y = next_move
                _score = get_score(next_move, player)
                # kiểm tra trên 4 đường nếu tổng đủ hơn 5, nếu bị chặn bởi opponent
                # hoặc ra khỏi board sẽ ngừng đếm
                if key == 'up_left' and  continue_move[0]:
                    if board[x,y] != player:
                        if score_every_direc['up_left'] != 5:
                            if board[x,y] == opponent or (is_in_board([x-1,y-1]) and board[x-1,y-1] == opponent):
                                score_every_direc['up_left']-=1
                        continue_move[0] = False
                    else:
                        score_every_direc['up_left']+=1
```

Thân hàm tính qua các hướng khác nhau …..

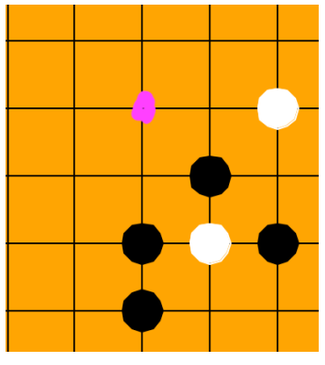
```python
                elif key == 'left' and continue_move[7]:
                    if board[x,y] != player:
                        if score_every_direc['left'] != 5:
                            if board[x,y] == opponent or (is_in_board([x,y-1]) and board[x,y-1] == opponent):
                                score_every_direc['left']-=1
                        continue_move[7] = False
                    else:
                        score_every_direc['left']+=1
    A_score = v_score(score_every_direc)
    if A_score[5] > 0: _score= 6
    elif A_score[4] > 0: _score= 5
    elif A_score[3] >= 2:_score= 4
    elif A_score[3] == 1:
        _score= 3
        for key, item in score_every_direc.items():
            if item == 3:
                if key == 'up' and score_every_direc['down'] >=2: _score = 6
                if key == 'down_right' and score_every_direc['up_left'] >=2: _score = 6
                if key == 'left' and score_every_direc['right'] >=2: _score = 6
                if key == 'down_left' and score_every_direc['up_right'] >=2: _score = 6
                if key == 'down' and score_every_direc['up'] >=2: _score = 6
                if key == 'up_left' and score_every_direc['down_right'] >=2: _score = 6
                if key == 'right' and score_every_direc['left'] >=2: _score = 6
                if key == 'up_right' and score_every_direc['down_left'] >=2: _score = 6
    elif A_score[2] > 0: _score= 2
    else: _score= 0
    if is_min_player: return -(_score)
    return _score
```

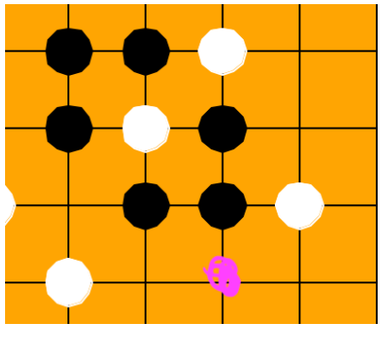
- Hàm này là hàm tính điểm, hàm này sẽ đự đoán điểm nếu chúng ta đánh vào nước đi đó, nên nó sẽ đánh thử để tính điểm
- Từ điểm đang xét tụi em tính ra 8 hướng, nếu hướng đó có:

➡ 5 thì điểm trả về là 6

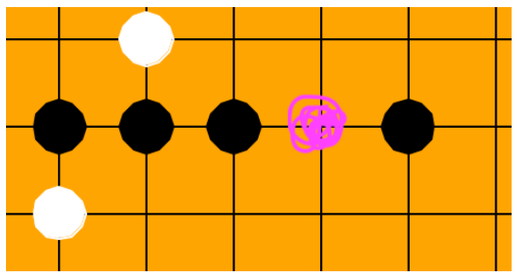
➡ 4 thì điểm trả về là 5

➡ nếu có 2 hướng đạt 3 điểm: 4





➡ nếu có 1 hướng 3, thì kiểm tra hướng còn lại, điểm đối ngược với hướng đó có lớn hơn 2 không nếu lớn hơn thì trả về 6 điểm, nếu không thì trả về 3 điểm



➡ nếu hướng đó có nhiều hướng đi 2 điểm, thì trả về 1

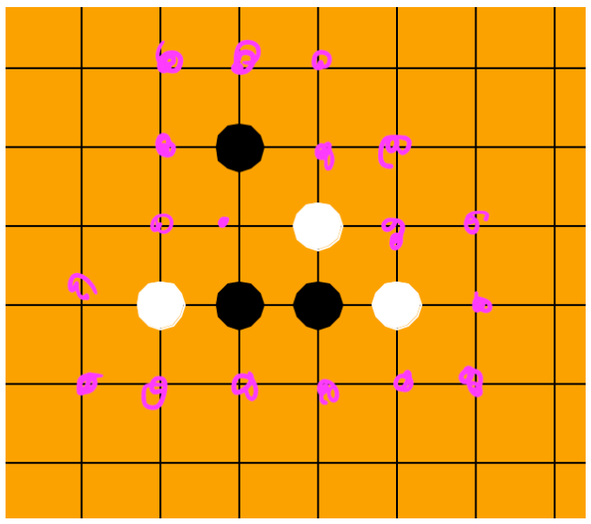
➡ còn ngược lại không tính điểm trả về 0

- Tuỳ vào người chơi chúng ta trả về điểm âm hay điểm dương

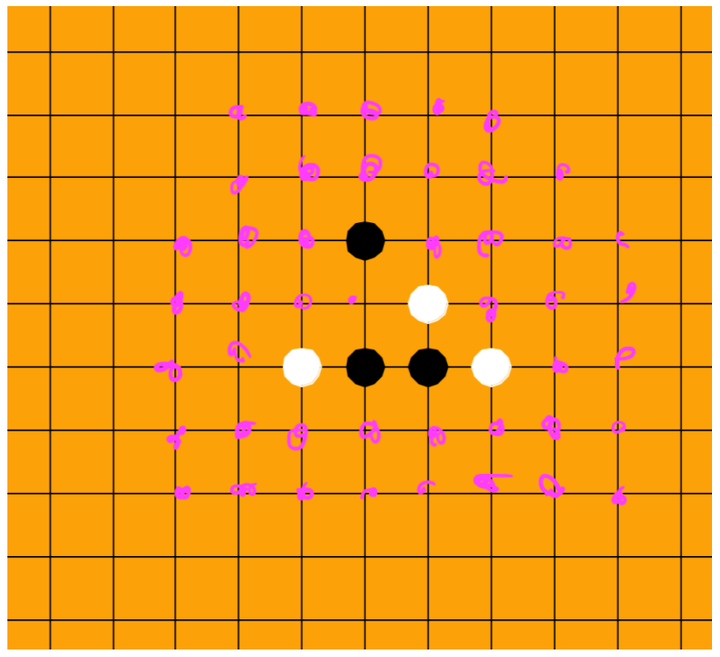
- Hàm này tìm các nước đi xung quanh có thể đi cách các nước đã đánh num đơn vị, ví dụ

```
# tìm trên map coi vị trí nào còn trống và gần ngươi chơi num đơn vị, vì không nên tìm hết map (rất mất time)
def find_index_near_to_player(num):
    index_empty = []
    moves = []
    for row in range(len(board)):
        for col in range(len(board)):
            if board[row,col] != 0:
                moves.append([row, col])
    for m in moves:
        x, y = m
        if x-num < 0: x1 = 0
        else: x1=x-num
        if y-num < 0: y1 = 0
        else: y1=y-num

        if x+num > LENGTH-1: x2 = LENGTH-1
        else: x2 = x+num
        if y+num > LENGTH-1: y2 = LENGTH-1
        else: y2 = y+num

        for i in range(x1,x2):
            for j in range(y1, y2):
                if [i,j] not in index_empty and board[i,j] == 0:
                    index_empty.append([i,j])
    return index_empty
```

- NUM = 1



- NUM = 2



- Hàm Minmax Search

```
def minmaxSearch(board, cur_move ,is_min_player, depth=2):
#     global board
    _board = np.copy(board)
    i,j = cur_move
    if is_min_player:
        _board[i,j] = PLAYER_MIN
    else:
        _board[i,j] = PLAYER_MAX
    score = call_score(_board,cur_move,is_min_player)
    if depth == 0:
        return score
    row, col = cur_move
    values = []
    all_move = get_possible_move_to_cal_score(cur_move,_board,is_min_player)
    flag = False
    if is_min_player:
        best_val = -INF
        for key, moves in all_move.items():
            for next_move in moves:
                x, y = next_move
                _board = np.copy(board)
                _board[x,y] = PLAYER_MAX
                result = minmaxSearch(_board, next_move , False, depth-1)
                best_val = max(result, best_val)
    else:
        best_val = +INF
        for key, moves in all_move.items():
            for next_move in moves:
                x, y = next_move
                _board = np.copy(board)
                _board[x ,y] = PLAYER_MIN
                result = minmaxSearch(_board, next_move , True, depth-1)
                best_val = min(result, best_val)
    return best_val
```

- Hàm Alpha Beta

```
INF = 10e9
def alpha_beta(board=None, cur_move=None ,is_min_player=True, depth=2, alpha=-1000, beta=1000):
    _board = np.copy(board)
    i,j = cur_move
    if is_min_player:
        _board[i,j] = PLAYER_MIN
    else:
        _board[i,j] = PLAYER_MAX
    score = call_score(_board,cur_move,is_min_player)
    if depth == 0:
        return score
    row, col = cur_move
    values = []
    all_move = get_possible_move_to_cal_score(cur_move,_board,is_min_player)
    if len(all_move) == 0:
        return score
    if is_min_player:
        best_val = -INF
        for key, moves in all_move.items():
            for next_move in moves:
                x, y = next_move
                _board = np.copy(board)
                _board[x,y] = PLAYER_MAX
                result = alpha_beta(_board, next_move , False, depth-1, alpha, beta)
                best_val = max(result, best_val)
                if beta <= alpha:
                    return best_val
                alpha = max(alpha, result)
    else:
        best_val = +INF
        for key, moves in all_move.items():
            for next_move in moves:
                x, y = next_move
                _board = np.copy(board)
                _board[x ,y] = PLAYER_MIN
                result = alpha_beta(_board, next_move , True, depth-1, alpha, beta)
                best_val = min(result, best_val)
                if beta <= alpha:
                    return best_val
                beta = min(beta, result)
    return best_val
```

- Hàm play() hàm chơi dựa vào ngừoi chơi đang chơi là ai, thuật toán đang sử đụng là cái nào để tìm hết nước đi của cả 2 người chơi min và max
- Dựa vào người đang play để biết đối thủ có nước đi tốt hơn hay không, nếu tốt hơn thì sẽ tấn công ngược lại sẽ phòng thủ

```
# hàm đi của mỗi nước
def play(PLAYER_FIRST, HUMAN=False, move=[-1,-1], algo="MINMAXSEARCH"):
    if 0 not in board:
        return 0
    if 2 not in board:
        next_move = [int((LENGTH-1)/2),int((LENGTH-1)/2)]
        if board[int((LENGTH-1)/2),int((LENGTH-1)/2)] == 1:
            next_move = [int((LENGTH-1)/2)+1,int((LENGTH-1)/2)]
        row, col = next_move
        board[row,col] = PLAYER_MAX
        return PLAYER_MAX, [row,col]
    if PLAYER_FIRST == PLAYER_MIN:
        value = 1
        is_min_player = True
        next_player = False
    else:
        is_min_player = False
        next_player = True
        value = 2

    print_board(board)
    _PLAYER_MOVE_ATTACK = {}
    _PLAYER_MOVE = {}

    index_near_is_empty = find_index_near_to_player(is_min_player,3)
    print(index_near_is_empty)
    if algo == "MINMAXSEARCH":
        for index in index_near_is_empty:
            row, col = index
            _board = np.copy(board)
            _PLAYER_MOVE[(row,col)] = minimaxSearch(_board, [row,col] , is_min_player,2)
            _PLAYER_MOVE_ATTACK[(row,col)] = minimaxSearch(_board, [row,col] ,next_player,2)
    elif algo == "ALPHABETA":
        for index in index_near_is_empty:
            row, col = index
            _board = np.copy(board)
            _PLAYER_MOVE[(row,col)] = alpha_beta(_board, [row,col] , is_min_player,2,-1000,1000)
            _PLAYER_MOVE_ATTACK[(row,col)] =alpha_beta(_board, [row,col] ,next_player,2,-1000,1000)
```

- min player chính là máy, người chơi chính là max
  - ◉ _PLAYER_MOVE là của máy, điểm của _PLAYER_MOVE là điểm âm,
  - ◉ _PLAYER_MOVE_ATTACK là của người chơi, điểm của_PLAYER_MOVE_ATTACK là điểm dương
- Các trường hợp tấn công:
  - ▸ Nếu _PLAYER_MOVE mà có điểm lớn hơn 4, tức nó khả thi để win
  - ▸ Nếu _PLAYER_MOVE mà có điểm nhỏ hơn 3, mà người chơi tức _PLAYER_MOVE_ATTACK có khả năng win thì nó sẽ tấn công vào các nước đó để chặn người chơi
- ▸ Các trường hợp phòng thủ: ngược lại các trường hợp trên

```
    if is_min_player:
        min_key = min(_PLAYER_MOVE, key=lambda k: _PLAYER_MOVE[k])
        max_key = max(_PLAYER_MOVE_ATTACK, key=lambda k: _PLAYER_MOVE_ATTACK[k])
        move_filter_attack = (dict(filter(lambda x: x[1] == _PLAYER_MOVE_ATTACK[max_key], _PLAYER_MOVE_ATTACK.items())))
        move_filter = (dict(filter(lambda x: x[1] == _PLAYER_MOVE[min_key], _PLAYER_MOVE.items())))
        print(move_filter)
        print(move_filter_attack)
        # win
        if abs(_PLAYER_MOVE[min_key]) >= 4:
            print("DEFENSE")
            if len(move_filter) > 3:
                next_move = random.choice(list(move_filter.keys()))
            else:
                next_move = min(_PLAYER_MOVE.keys(),key=lambda x: _PLAYER_MOVE[x])
            print("Machine 1: ",next_move,move_filter[next_move])
        elif abs(_PLAYER_MOVE_ATTACK[max_key]) > 3 or abs(_PLAYER_MOVE[min_key]) <= 3: # chưa cần phòng thủ, tấn công
            print("ATTACK")
            if len(move_filter_attack) > 1:
                next_move = random.choice(list(move_filter_attack.keys()))
            else:
                next_move = max(_PLAYER_MOVE_ATTACK.keys(),key=lambda x: _PLAYER_MOVE_ATTACK[x])
            print("Machine 1: ",next_move,move_filter_attack[next_move])
        else:
            print("DEFENSE")
            if len(move_filter) > 3:
                next_move = random.choice(list(move_filter.keys()))
            else:
                next_move = min(_PLAYER_MOVE.keys(),key=lambda x: _PLAYER_MOVE[x])
            print("Machine 1: ",next_move,move_filter[next_move])
        row, col = next_move
        board[row,col] = PLAYER_MIN
        if is_win(board, [row,col], PLAYER_MIN):
            return 0, [row,col]
        return PLAYER_MIN, [row,col]
    elif HUMAN:
        max_key = max(_PLAYER_MOVE, key=lambda k: _PLAYER_MOVE[k])
        move_filter = (dict(filter(lambda x: x[1] == _PLAYER_MOVE[max_key], _PLAYER_MOVE.items())))
        if len(move_filter) > 1:
            next_move = random.choice(list(move_filter.keys()))
        else:
            next_move = max(_PLAYER_MOVE.keys(),key=lambda x: _PLAYER_MOVE[x])
        print("Recommend: ",next_move)
        board[move[0],move[1]] = PLAYER_MAX

        if is_win(board, [row,col], PLAYER_MAX):
            return 0, [row,col]
        return PLAYER_MAX, [row,col]
```

3. Reference

Giải Thuật Cắt Tỉa Alpha-beta

https://www.stdio.vn/articles/giai-thuat-cat-tia-alpha-beta-564

Giải Thuật Minimax Search

https://vi.wikipedia.org/wiki/Minimax#Ti%C3%AAu_chu%E1%BA%A9n_Minimax_trong_l%C3%BD_thuy%E1%BA%BFt_quy%E1%BA%BFt_%C4%91%E1%BB%8Bnh_th%E1%BB%91ng_k%C3%AA