ĐẠI HỌC QUỐC GIA VIỆT NAM THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

GAME PLAYING 2 PLAYER

Giảng viên hướng dẫn: Vương Bá Thịnh
Sinh viên thực hiện: Nguyễn Đắc Hoàng Phú - 2010514
Đoàn Trần Cao Trí - 2010733
Du Thành Đạt - 2010206

Thành phố Hồ Chí Minh, 03/2022

Contents

1 Giới Thiệu

Bài tập lớn bao quanh việc hiện thực và áp dụng giải thuật turnbased nhằm áp dụng vào các chiến thuật trong các trò chơi đối kháng, điển hình là các trò chơi đánh cờ, nơi mà mỗi đối thủ đều có tính toán riêng để đạt được những nước đi có lợi nhất. Một số giải thuật turnbased đã biết trước đó bao gồm:

+ **Minimax-algorithm:** Minimax là giải thuật đệ quy lựa chọn bước đi kế tiếp trong một trò chơi có hai người bằng cách định giá trị cho các Node trên cây trò chơi sau đó tìm node có giá trị phù hợp để đi bước tiếp theo dựa vào hàm lượng giác.

+ **Alpha-beta pruning:** Giải thuật này thường sử dụng chung với thuật toán tìm kiếm Minimax nhằm hỗ trợ giảm bớt các không gian trạng thái trong cây trò chơi, giúp thuật toán Minimax có thể tìm kiếm sâu và nhanh hơn. Giải thuật cắt tỉa Alpha-beta có nguyên tắc đơn giản "Nếu biết là trường hợp xấu thì không cần phải xét thêm".

+ **Monte-Carlo Search:** Có thể xây dựng hàm lượng giá hành động tốt để đánh giá các nước đi khả thi cho máy tính có thể lựa chọn được nước đi tốt nhất dựa vào một khoảng thời gian ngắn, phương pháp này còn đưa ra các thống kê để lựa chọn đặc trưng nhằm có một chiến thuật chơi game tốt nhất.

Loại trò chơi đối kháng 2 người chơi mà nhóm chọn là game cờ Isolation (hay Isola)

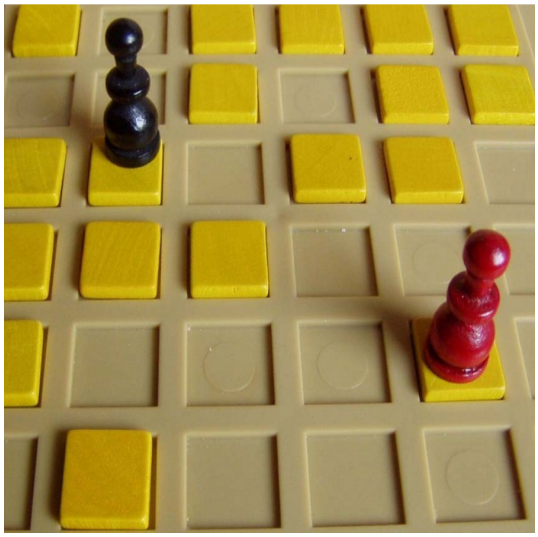


Figure 1: *Một số hình ảnh từ trò chơi*

Luật chơi

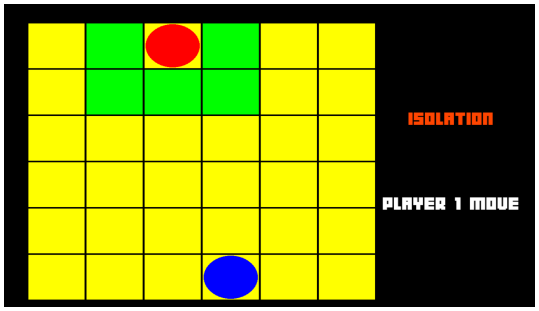
+ Cấu hình trò chơi là bàn cờ với kích thước 6*6, chỉ có hai quân cờ duy nhất có màu khác nhau (thường là màu xanh và màu đỏ).

+ Hai quân cờ có vị trí đối diện nhau và ở góc bàn cờ, có nhiều biến thể nước đi, nhóm chọn cách đi mỗi con cờ như quân vua trong trò cờ vua, tức là nếu quân cờ đang ở ô (x,y) thì nó có thể di chuyển 1 trong 8 ô bao quanh nó gồm (x+1,y), (x-1,y), (x,y-1), (x,y+1), (x+1,y-1), (x+1,y+1), (x-1,y-1), (x-1,y+1).

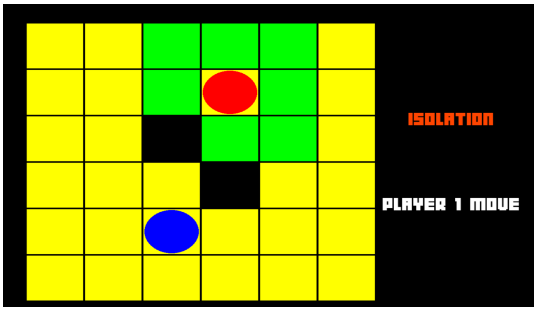
+ Ta định nghĩa một vị trí bị đánh dấu tức là vị trí đó sẽ không di chuyển được, tức là các quân cờ sẽ không có quyền di chuyển vào ô đó. Sau mỗi lượt di chuyển quân cờ, ta có quyền chọn một vị trí chưa bị đánh dấu và đánh dấu nó (ngoại trừ 2 ô đang đứng của 2 quân cờ).

+ Trò chơi sẽ kết thúc khi có quân cờ không thể di chuyển một nước nào khác, tức là các ô bao xung quanh quân cờ đó đều đã bị đánh dấu.

2 Giao diện trò chơi



(a) *Giao diện bắt đầu trò chơi*



(b) *Quá trình diễn ra trò chơi*

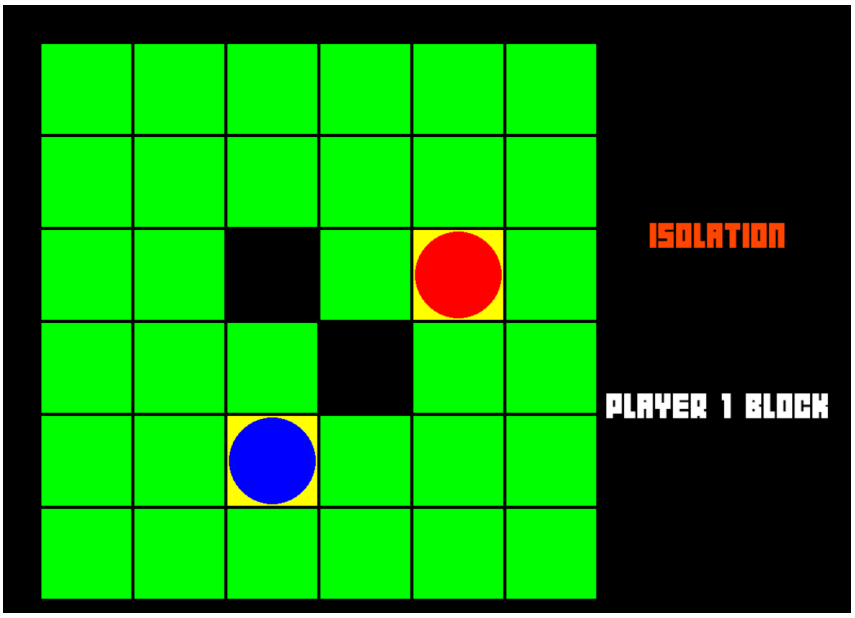


Figure 3: *Chỉ dẫn những nước đi hợp lệ*

Mô tả giao diện

- Bên trái theo hướng người xem là bảng trò chơi (Ở đây là 6x6).
- Hình tròn màu **Đỏ** là người chơi 1
- Hình tròn màu **Xanh** là người chơi 2
- Ô vuông màu **Vàng** là ô vuông đang trống
- Ô vuông màu **Xanh** là ô vuông hợp lệ ngay tại lượt đi đó.
- Ô vuông màu **Đen** là ô vuông đã bị chặn theo như trong luật trò chơi.
- Bên phải người xem là tên trò chơi **ISOLATION**
- Phía dưới là thông tin về lượt chơi hiện tại hoặc kết quả chung cuộc.

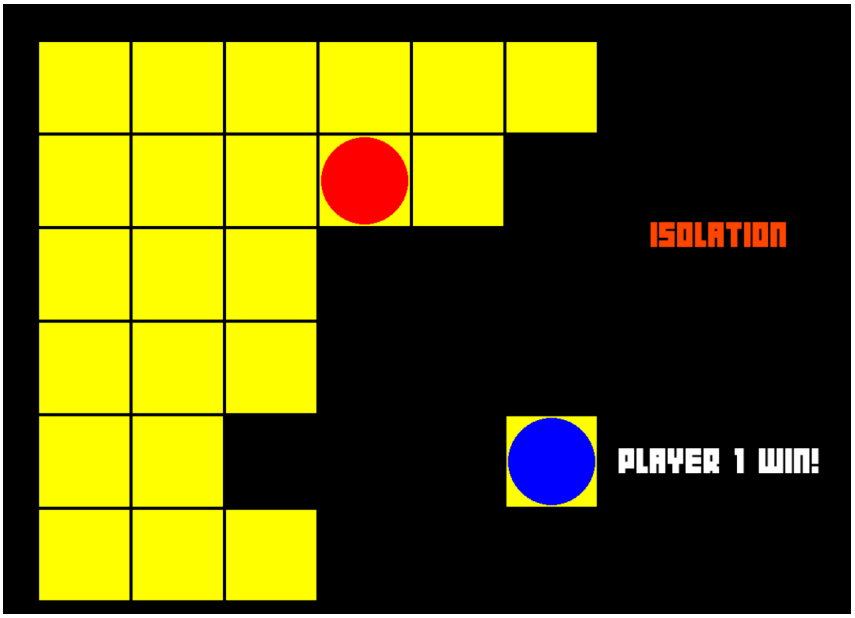


Figure 4: *Kết quả chung cuộc, người chơi 1 (Đỏ) chiến thắng*

3 OOP design

Nhóm chọn cách lập trình hướng đối tượng để hiện thực bài toán. Dưới đây là class diagram và các interface được sử dụng để định nghĩa lên trò chơi Isolation.

3.1 Interface game

3.1.1 Environment.py

Class này được sử dụng để định nghĩa môi trường để các agent thực hiện theo các thuật toán Minimax hay cả là con người để tương tác với nhau.

```
class BaseEnvironment:
    __metaclass__ = ABCMeta

    @abstractmethod
    def __init__(self,depth,path={}):
        pass

    @abstractmethod
    def env_act(self):
        pass

    @abstractmethod
    def reset(self,player,depth,path):
        pass

    @abstractmethod
    def check_win(self):
        pass

    @abstractmethod
    def step(self,action):
        pass
```

3.1.2 MinimaxAlgorithm.py

Class này để định nghĩa lên các phương thức được sử dụng cho thuật toán Minimax được sử dụng làm các agent heuristic.

```
class Minimax(object):
    __metaclass__ = ABCMeta

    @abstractmethod
    def __init__(self,depth,board = None):
        pass

    @abstractmethod
    def get_all_nearby_cell(self,x,y):
        pass

    @abstractmethod
    def get_all_empty_cell(self,board,x,y):
        pass

    @abstractmethod
    def move(self,board,player):
        pass

    @abstractmethod
    def aimove(self,board,player):
        pass

    @abstractmethod
    def print_board(self,board):
        pass

    @abstractmethod
    def evaluate_board(self,board,agent_flag):
        pass

    @abstractmethod
    def in_board(self,x,y):
        pass

    @abstractmethod
    def checkEnd(self,board,agent_flag):
        pass

    @abstractmethod
    def clone_board(self,board):
        pass

    @abstractmethod
    def minimax(self,board,player,maximizer,alpha,beta,depth):
        pass

    @abstractmethod
    def get_player_piece(self,board,agent_flag):
        pass

    @abstractmethod
    def strategy1_check_point(self,board,componet_flag,cell):
        pass

    @abstractmethod
    def strategy1_block_move(self,board,component_flag,component_cell):
```

```
        pass

    @abstractmethod
    def player_move(self,board,player):
        pass


    @abstractmethod
    def move_piece(self,board,player_flag,old_cell,new_cell):
        pass
```

3.1.3 IsolationEnv.py

Các class này được tạo ra để hiện thực các chức năng đánh giữa người với người, người với minimax, người với igen và minimax với minimax.

```
class Env(BaseEnvironment):
    def __init__(self,depth = 0,path={}):
        pass

    def env_act(self):
        pass

    def print_board(self):
        pass

    def reset(self,player,depth,path):
        pass

    def check_win(self):
        pass

    def step(self):
        pass
```

3.2 Class diagram

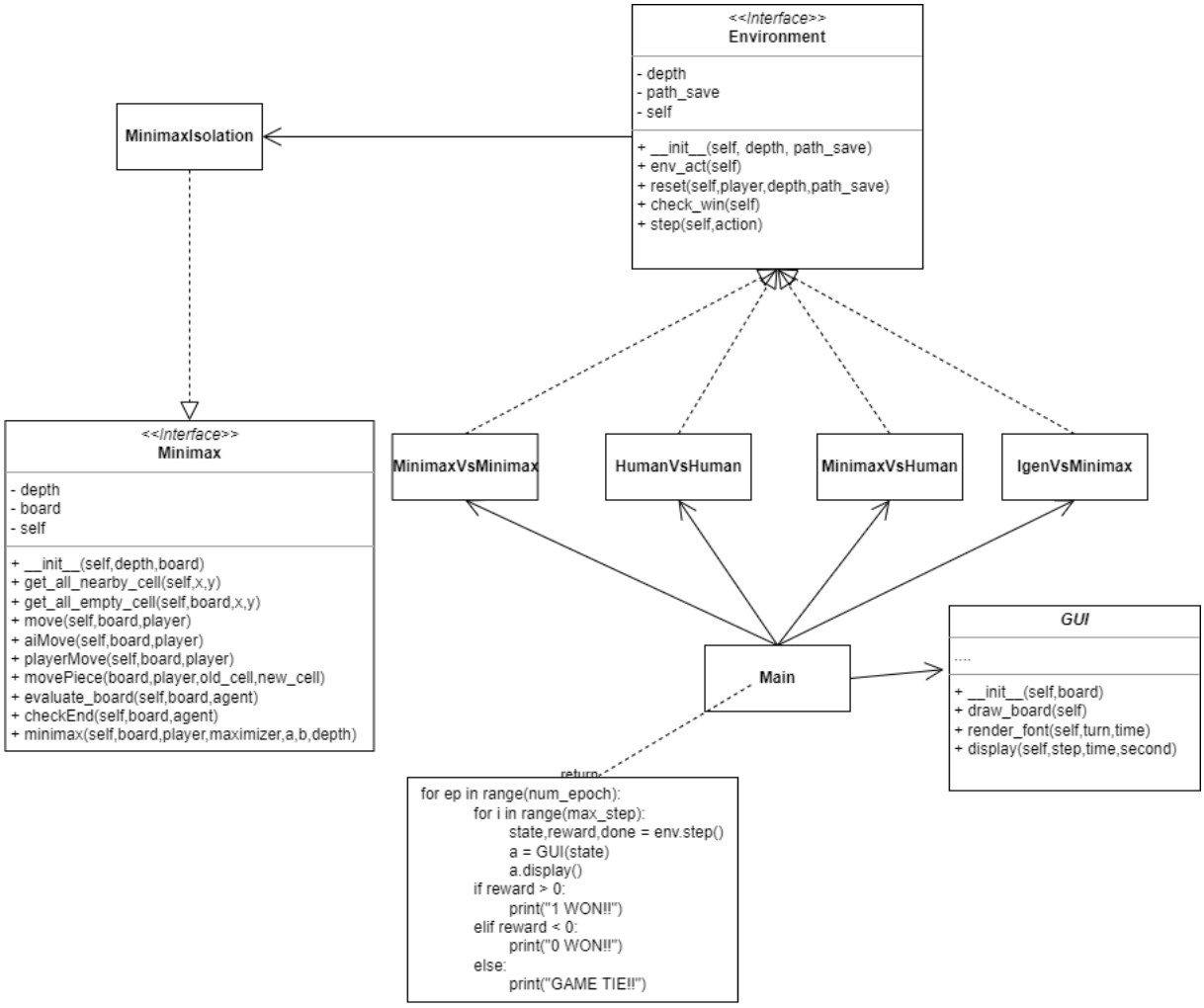


Figure 5: *Sơ đồ UML để hiện thực Isolation*

4 Giải thuật Minimax sử dụng cho bài toán

Giải thuật Minimax thực hiện theo cơ chế cắt alpha-beta pruning cộng thêm iterative-deeping nhằm làm giảm xuống độ phức tạp của không gian tìm kiếm.Code của thuật toán, được tối ưu và hiện thực như sau:

+ **Bước 1:** Đầu tiên thuật toán kiểm tra depth, nếu depth đã đạt được độ sâu mà ta đã đặt thì trả về giá trị tốt nhất trong từ trước đến mức đó.

+ **Bước 2:** Tiếp đến thuật toán kiểm tra xem, đối phương hay ta đã hết nước đi để trả về kết quả người thắng cuộc

+ **Bước 3:** Nếu đang là người maximizer, ta sẽ chọn giá trị lớn nhất từ trạng thái của cấu hình (giá trị này được định nghĩa là số ô trống xung quanh quân cờ đang lượt hiện tại), sau đó trả về chiến lược đi dựa vào giá trị tốt nhất đó, cùng nước đi đánh dấu ô không được đi, minimizer tương tự bằng cách chọn giá trị bé nhất.

+ **Bước 4:** Cập nhật lại giá trị alpha, beta và chuyển đến lược tiếp theo.

```
def minimax(self,board,player,maximizer,alpha,beta,depth):
    if depth == 0:
        value = self.evaluate_board(board,player)
```

```
        return (value,None,depth)
    # self.print_board(board)
    component_pieces = self.get_player_piece(board,-1 * player)
    player_pieces = self.get_player_piece(board,player)
    #winner
    print('component_pieces: ',component_pieces,len(component_pieces))
    if len(component_pieces) == 0:
        value = float('inf')
        return (value,None,depth)
    print('player_pieces: ',player_pieces,len(player_pieces))
    if len(player_pieces) == 0:
        print("here")
        value = float('-inf')
        return (value,None,depth)
    #maximizer <-> 1 -> player(1); 0 -> player(1);
    print("maximizer: ",maximizer)
    if maximizer == 1:
        best_value = float('-inf')
        best_move = None
        best_depth = float('-inf')
        # save_now_cell = None
        for cell in player_pieces:
            new_board = self.clone_board(board)
            # save_now_cell = self.pos_agent[player]
            cur = self.get_pos(new_board,player)
            self.move_piece(new_board,player,cur,cell)
            result = self.minimax(new_board,-player,0,alpha,beta,depth - 1)

            if best_value < result[0] or (best_value == result[0] and result[2] > best_depth):
                best_value = result[0]
                best_move = (cur,cell)
                best_depth = result[2]

            if result[0] >= beta:
                # self.pos_agent[player] = save_now_cell
                return (result[0],(cur,cell),best_depth)

            if alpha < result[0]:
                alpha = result[0]
        # self.pos_agent[player] = save_now_cell
        return (best_value,best_move,best_depth)
    else:
        best_value = float('inf')
        best_move = None
        best_depth = float('-inf')
        # save_now_cell = None
        for cell in player_pieces:
            new_board = self.clone_board(board)
            # save_now_cell = self.pos_agent[player]
            cur = self.get_pos(new_board, player)
            self.move_piece(new_board,player,cur,cell)
            result = self.minimax(new_board,-player,1,alpha,beta,depth - 1)

            if best_value > result[0] or (best_value == result[0] and result[2] > best_depth):
                best_value = result[0]
                best_move = (cur,cell)
                best_depth = result[2]

            if alpha >= result[0]:
                # self.pos_agent[player] = save_now_cell
                return (result[0],(cur,cell),best_depth)
            if beta > result[0]:
```

```
                beta = result[0]

        # self.pos_agent[player] = save_now_cell
        return (best_value,best_move,best_depth)
```

Chiến thuật để đánh dấu vị trí để các quân cờ không di chuyển vào vị trí này, nhóm sử dụng chiến thuật tham lam tức là dựa vào số ô có thể di chuyển của đối phương mà làm cho số ô đó ít nhất có thể, cụ thể cách hiện thực như sau:

+ Với mỗi ô có thể di chuyển được từ đối phương, ta lần lượt đánh dấu thử các ô đó và định nghĩa cách tính điểm cho ô đó với giá trị khởi tạo ban đầu bằng 0. Và xem nước đi nào tốt nhất cho đối phương và block ngay ô đó.

+ Đầu tiên, điểm được tính bằng cách nếu ô đó ngay rìa thì sẽ trừ đi 2 điểm (bởi vì các nước đi ngay rìa sẽ không tốt bằng các nước ở phía trong bàn cờ, điểm trừ này thiên hướng về sự không tốt đó).

+ Thứ hai với vị trí hiện tại của ô đặt quân cờ đối thủ, sẽ có 4 hướng theo 8 ô xung quanh là trên dưới trái phải, lúc đó kiểm tra xem các hướng với mỗi ô bị BLOCK, điểm sẽ cộng 1. Những ô ta định đánh dấu thuộc giao của 2 hướng bất kì mà bị BLOCK sẽ mang thiên hướng được nhiều điểm hơn.

+ Cuối cùng chọn ra ô có điểm cao nhất để rồi đánh dấu nó.

```
def strategy1_check_point(self,board,component_flag,cell):
    x,y = self.get_pos(board,component_flag)
    #left condition
    x_cell,y_cell = cell
    point = 0
    if x_cell == 0 or y_cell == 0:
        point = point - 2
    if x_cell == WIDTH - 1 or y_cell == WIDTH - 1:
        point = point - 2
    #left condition
    board[x_cell][y_cell] = BLOCK
    for label in DIRECTION_LABEL:
        direction_lst = DIRECTION[label]
        for direct in direction_lst:
            x_nei = x + direct[0]
            y_nei = y + direct[1]
            if self.in_board(x_nei,y_nei) and board[x_nei][y_nei] != FREE:
                point += 1

    board[x_cell][y_cell] = FREE
    return point
```

5 Hướng dẫn chạy code và trải nghiệm

5.1 2 người chơi

File chạy: **HumanVsHuman.py**

5.2 Chơi với máy

File chạy: **MinmaxVsHuman.py**

5.3 Xem máy tự động chơi

File chạy 1: **MinimaxVsMinimax**
Dữ liệu đầu vào là Độ sâu của cây Minimax bao gồm minDepth và maxDepth

- depth for maximizer: ?
- depth for minimizer: ?

File chạy 2: **MinimaxVsIgen**

- depth for maximizer: ?

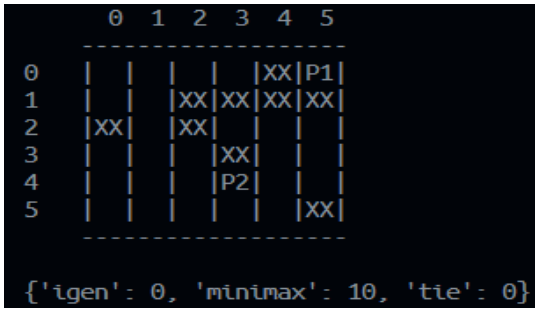
6 Kết quả

6.1 Agent chơi đúng luật:

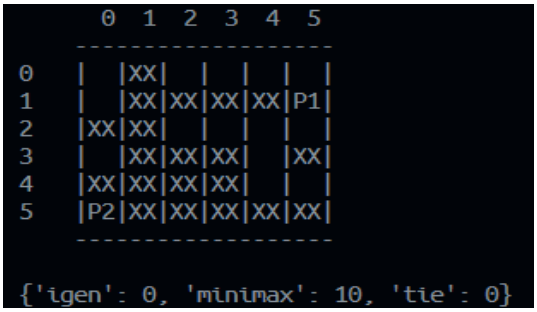
Bằng thực nghiệm và chơi thử nhiều lần trên giao diện với agentMinimax cũng như agentIgen, nhóm tự tin kiểm chứng được độ chính xác của các agent chơi đúng luật.

6.2 Về kết quả với Agent ngẫu nhiên:

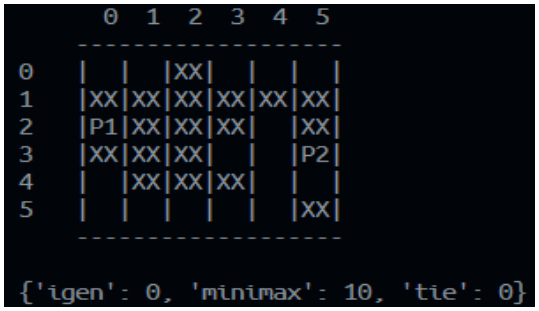
Nhóm thực hiện viết một chương trình tournament.py với sự tham gia của 2 agent igen và Minimax, trong đó lượt đi trước sẽ được random bất kì giữa 2 agent, tổng cộng đấu 10 ván, nhóm chạy 8 lần file và kết luận được rằng agentMinimax thông minh và mỗi lần chạy vậy đều thắng hết 10/10 ván.



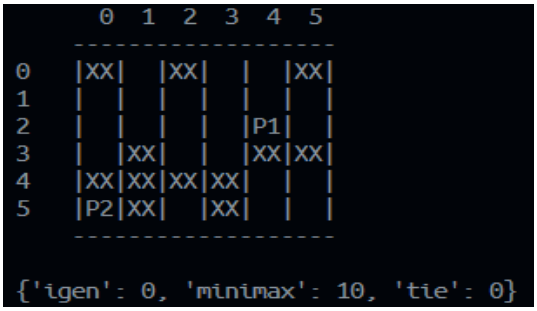
(a) *Lần chạy tournament.py đầu tiên*



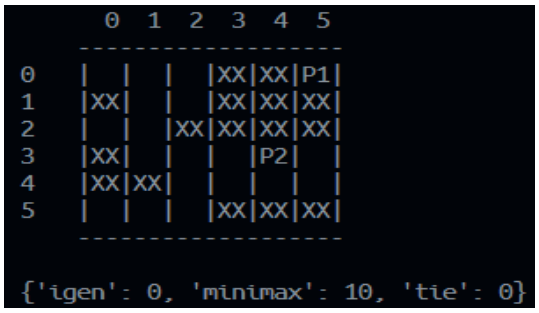
(b) *Lần 2*



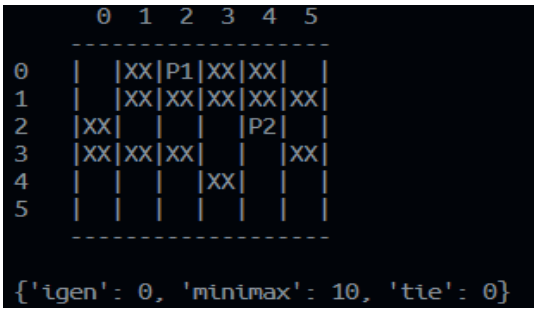
(a) *Lần 3*



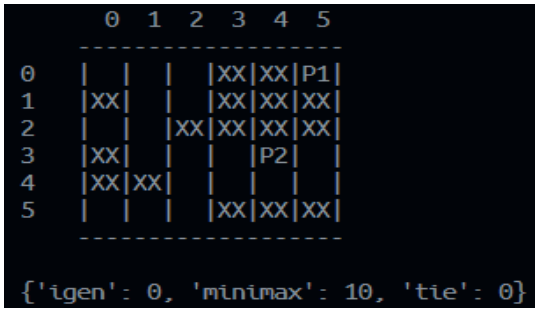
(b) *Lần 4*



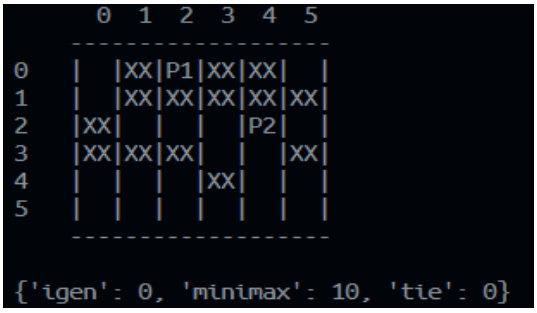
(a) *Lần 5*



(b) *Lần 6*



(a) *Lần 7*



(b) *Lần 8*

6.3 Về đánh giá cấp độ của agent so với con người:

+ **Phân theo độ sâu của cây tìm kiếm:** Ta có thể tùy chỉnh độ sâu của cây tìm kiếm minimax để giúp cho agent thông minh hơn (thời gian tìm kiếm sẽ lâu hơn) và ngược lại

+ **Phân theo thuật toán sử dụng cho agent:** Nhóm chia ra 3 cấp độ thông minh nhất là Minimax, trung bình là Igen (chọn vị trí di chuyển bất kì) với theo chiến thuật chọn ô đánh dấu như trên và dở nhất là Igen với chọn ô đánh dấu bất kì.

7 Hướng đi kế tiếp

+ Nhóm chọn game Isolation ngoài thử sức lập trình các agent cơ bản theo giải thuật turnbased để có thể đấu người thì mong muốn tiếp tục phát triển nhiều agent thông minh hơn nữa để phục vụ cho bài tập lớn số 3.

+ Tiếp cận nhiều thuật toán liên quan AI, reinforcement learning để giúp cho agent tiết kiệm được không gian tìm kiếm cũng như cho một giải pháp nhanh hơn trong việc quyết định ra nước đi kế tiếp

+ Một số giải thuật nhóm sẽ dự định thực hiện: TD-learning, Q-learning và DeepLearning với một số cơ chế như epsilon-greedy và Experience-replay,..

+ Ngoài ra Nhóm cũng sẽ viết thêm các class agent, class model vào class-diagram hiện có để mở rộng thêm mô hình.

8 Tài liệu tham khảo

- Isolation Minimax Gaming
- Tutorial Isolation
- alpha-beta pruning
- Minimax algorithm implementation
- Monte Carlo Tree Search