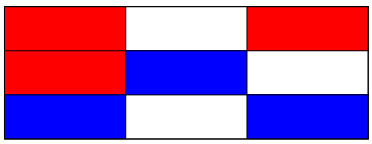
project1

20171300 지용환

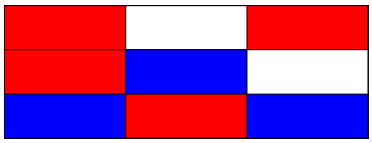
GIT: https://github.com/YONGYONGA/guaZ.git (20171300_pj1.py)

1. MINIMAX

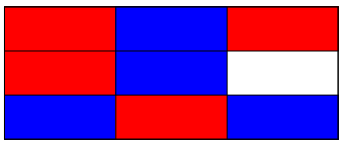
First, I explain about minimax algorithm. To summarize this algorithm in one line, it simply selects the most advantageous number from the number of cases. For example, if there is a case where the computer can win from the computer's point of view, it is selected. Let's think about TTT game. Blue color is player and red color is computer.

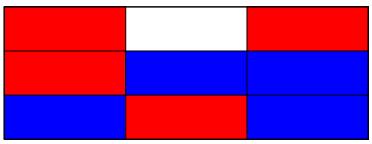


now, it's red turn. I,e computer turn. Here, there are 3 possible numbers for computer.

 (This is i case)

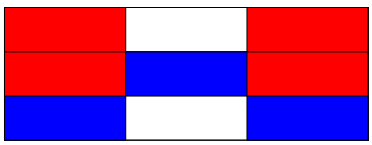
Suppose the computer puts it this way. Then, the player has 3 cases.

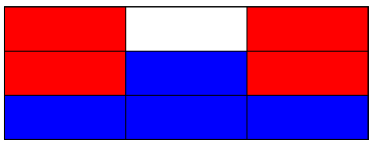


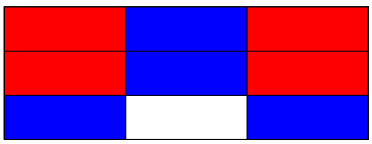


Let's call them 1, 2 in order. Since now is red turn, 1 is draw and 2 is red's win. We'll give -10 if Red wins, 10 if Blue wins, and 0 points if draw in each case. Hence 1 is 0 point and 2 is -10 point.

Similary, if computer puts below way (This is ii)

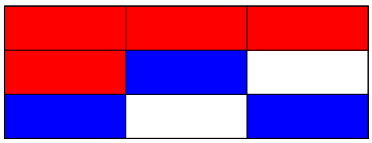
 2 case is possible, thas is



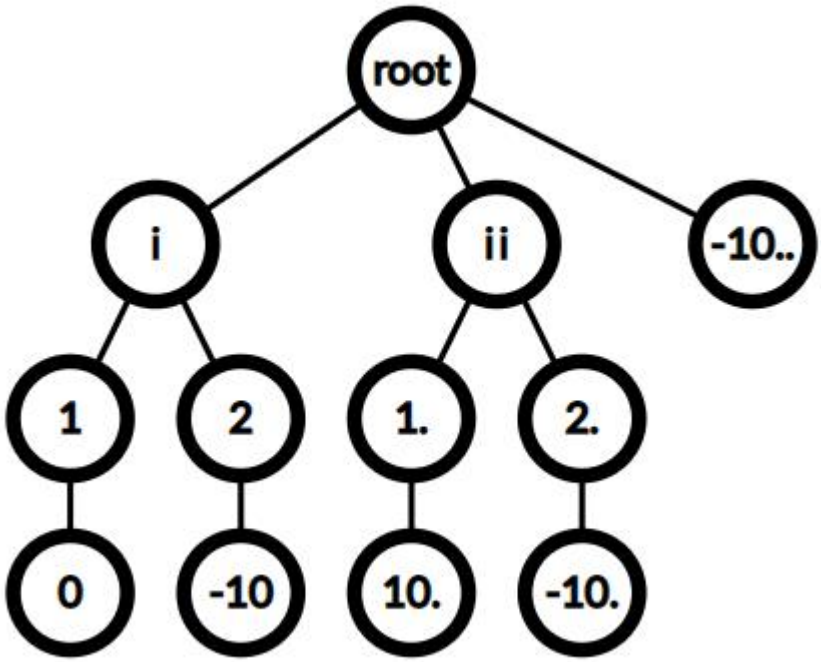


Then 1 is 10 point and 2 is -10 point.

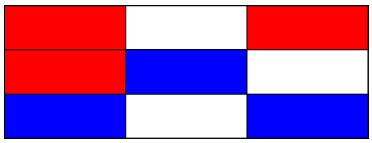
Finally in below case

 (this is iii)

it's -10 point. The game is already over, so there is no need to proceed further. Now let's organize this into a tree.



You can start from the bottom, and the bottom is the turn of red (computer). At this time, red has an advantage -score. That is, the minimum value is selected (but there is only one case, so it does not mean much). Anyway, at depth 3, it's the player's turn, and at each node, the maximum value is pushed up to the selection. I,e i get 0 score, and ii get 10 score. This means that the player has chosen the most advantageous case for himself, in i , ii cases. Then at depth, computer turn, and 0 10 -10 is possible. What will the computer choose? Naturally, the computer with the advantage of negative numbers chooses -10.

 In the first example, of course, it is the case of ending the game by selecting index 8.

In the TTT game, the game is so simple and calculate all possiblility, that

if you apply this algorithm, you can never beat the computer. Also, we almost always just draw because this game is easy. Then how about Go or Chess? Go and chess have a huge number of cases. In other words, if a computer can only calculate the number of cases, a person can never beat a computer. However, because of the large number of cases, even a computer cannot examine all cases. Therefore, several methods are used to tell the computer to do the best calculation. However, since it is not necessary in TTT, I will briefly explain it. The first method is to limit the depth. This method is simply to try to escape at the appropriate depth, because if you search too much and the depth of the tree gets too high, the computation becomes serious. The second way is to award additional points in addition to win, loss and draw. In TTt gaemes, it can calculate only using win, lose, draw(10,-10,0) since it is simple game. But in go and chess it's impossible, and give another point. For example if you catch the opponent's piece, get more points (of course, the better the piece you catch, the higher the score). The final method is pruning. I,e if it is a branch of a tree that is not needed, it is simply thrown away. This has an advantage in computational speed as it greatly reduces the number of cases to be calculated.
Now, let's apply this algorithm to the TTT created with pygame.

2. implement TTT
As explained, 10,-10,0 points for a win, loss, and draw respectively.

```
def findBestMove(board,MM) :
    bestVal = -1000
    bestMove = (-1, -1)
    ##MM=1인건 player turn. 즉 최대값이 필요
    ##MM=-1인건 computer turn 즉 최소값이 필요.
    # Traverse all cells, evaluate minimax function for
    # all empty cells. And return the cell with optimal
    # value.
    if(MM==1):
        bestVal = -1000
        for i in range(1,10):
            if board[i]==' ':
                board[i]=playerLetter
                moveVal=minimax(board,0,False)
                board[i]=' '
                if(moveVal>bestVal):
                    bestMove=i
                    bestVal=moveVal
        print("The value of the best Move is :", bestVal)
        print("at : ",bestMove)
        print()
    elif(MM==-1):
        bestVal=1000
        for i in range(1,10):
            if board[i]==' ':
                board[i]=computerLetter
                moveVal=minimax(board,0,True)
                board[i]=' '
                if(moveVal<bestVal):
                    bestMove=i
                    bestVal=moveVal
        print("The value of the best (com)Move is :", bestVal)
        print("at : ",bestMove)
        print()
    return bestMove
```

This function is a function that finds the optimal position. In other words, it finds the position where it is absolutely held. If MM=1 then it find maximum, and if MM=-1 it find minimum(which mean computer turn.) Only MM=1 will be explained. The bestVal is value for find maximum value. Using for loop, if gives playerLetter in current board (i.e. current game situation) and find maxinum using minimax. And restore the board and find maximum. In other words, it searches all possible cases in the current game situation and finds the max value.

Below is minimax function.

```
def minimax(board, depth, isMax) :
    score = evaluate(board)
    if (score == 10) :
        return score
    if (score == -10) :
        return score
    if (isBoardFull(board) == True) :
        return 0
        #아직 max값 못찾음, 현재 정보에서 재귀적으로 호출 ㄱ. 대신 다음은
    if (isMax) :
        best = -1000
        for i in range(1,10):
            if(board[i]==' '):
                board[i]=playerLetter
                best=max(best,minimax(board,depth+1,not isMax))
                board[i]=' '
        return best
    else :
        best = 1000
        for i in range(1,10):
            if(board[i]==' '):
                board[i]=computerLetter
                best=min(best,minimax(board,depth+1,not isMax))
                board[i]=' '
        return best
```

Using score(This is just a function that uses isWinner to return 10 points if the player wins and -10 points if the player loses.) and isFullboard, it return point if games is end(win or lose or draw), Otherwise, use the recursive method. Similar to the findbestmove function above, it finds possible cases for a given board. At first, isMAx is false, so it goes into the else statement below, and here it finds and returns the min value. At this time, the return value of minimax is included again in finding min. This is the recursive way. In other words, go to the end (until the end of the game) get the result and return it, and as explained in minimax algorithm, get the min, max value for each turn and pass it to the parent node. In the case of the example given at the beginning, it returns -10 in the end. This recursive method makes it very simple to apply the minimax algorithm to simple games. Now we need to apply this.

```
while True:
    if(gamerealend):
        break
    if(initial!=0 and gameend==False and turn=='player' and M==0):
        M=1
        bee=findBestMove(theBoard,M)
        #print(bee)
        xe=(abs(int((bee-1)/3)-2))
        ye=(bee-1)%3
        #print((xx,yy))
        text = font1.render(playerLetter, True, (255,192,203))
        screen.blit(text, [int(ye)*140,int(xe)*140])
        pygame.display.flip()
```

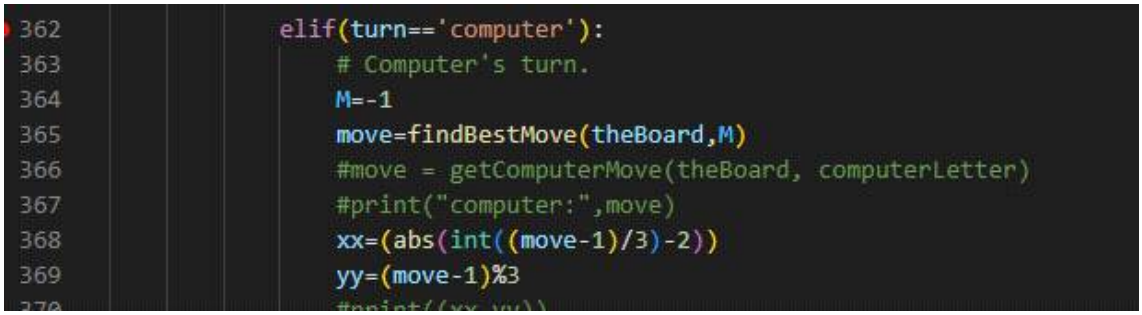
M is global value, initial value is 0. If player turn, If it is the player's turn, the best position is returned through the findBestMove function. If so, the above code converts this location and draws it on the screen.

```
                    gamerealend=True
        elif event.type==pygame.MOUSEBUTTONDOWN:
            if(initial!=0 and gameend==False and turn=='player'):
                pos = pygame.mouse.get_pos()
                mouse_x = pos[0]
                mouse_y = pos[1]
                if(check_mouse(mouse_x,mouse_y,theBoard)):
                    M=-1
                    if(bee!=-1):
                        #추천했던 위치 안보이게 초기화
                        clear_screen(bee)
                        bee=-1
                    mouse_c=True
```

Instead, since the player does not unconditionally press the recommended position, when a click event occurs in the mouse event part, erase the recommended position above and add a part that changes M=-1.

```
def clear_screen(location):
    xe=(abs(int((location-1)/3)-2))
    ye=(location-1)%3
    ys=int(ye)*140
    xs=int(xe)*140
    pygame.draw.rect(screen,WHITE,(ys+2,xs+2,135,135))
    pygame.display.flip()
```

clear_screen function is a function that erases the recommended location, that it is a method of simply overpainting the recommended location with a white square.



```
362             elif(turn=='computer'):
363                 # Computer's turn.
364                 M=-1
365                 move=findBestMove(theBoard,M)
366                 #move = getComputerMove(theBoard, computerLetter)
367                 #print("computer:",move)
368                 xx=(abs(int((move-1)/3)-2))
369                 yy=(move-1)%3
```

And if computer turn, it is simple. Just replace getComputerMove to findBestMove. Then move get best location using MINIMAX algorithm and after that, draw it in the original way. Surely, M becomes 0. Since now is player turn. Now, you can never beat the computer, and if you don't follow the recommended positions, you're more likely to lose.