TIC TAC TOE

Poojari Venkatesh
S20180010135
Sec-B

installing the required libraries

In [1]:
```
# using pygame 1.9.6
!pip install pygame
```

```
Requirement already satisfied: pygame in /home/venkatesh/anaconda3/lib/python3.7/site-packages (1.9.6)
```

importing libraries

In [2]:
```
from math import inf as infinity
from random import choice
import time
import numpy as np
from itertools import permutations
from collections import Counter

# for game visualization
import tkinter as tk
from PIL import Image, ImageTk
import pygame

# for sounds
from pygame import mixer
```

```
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html (https://www.pygame.org/contribute.html)
```

normal minimax algorithm

In [3]:

```python
def minimax(state, depth, player):
    if player == COMP:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]

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

    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

        if player == COMP:
            if score[2] > best[2]:
                best = score  # max value
        else:
            if score[2] < best[2]:
                best = score  # min value

    return best
```

minimax with alpha-beta pruning algorithm

In [4]:

```python
def minimax_alpha_beta(boardstate, depth, player, alpha, beta):
    alpha_new = alpha
    beta_new = beta
    if player == COMP:
        best = [-1, -1, -infinity, alpha, beta]
    else:
        best = [-1, -1, +infinity, alpha, beta]

    if depth == 0 or game_over(boardstate):
        score = evaluate(boardstate)
        return [-1, -1, score, score, score]

    for cell in empty_cells(boardstate):
        x, y = cell[0], cell[1]
        boardstate[x][y] = player
        score = minimax_alpha_beta(boardstate, depth - 1, -player, alpha_new , beta_new )
        boardstate[x][y] = 0
        score[0], score[1] = x, y

        # max value
        if player == COMP:
            if score[2] > best[2]:
                best = score
            if score[2] > alpha_new:
                alpha_new = score[2]
        # min value
        else:
            if score[2] < best[2]:
                best = score
            if score[2] < beta_new:
                beta_new = score[2]
        #alpha beta
        if alpha_new >= beta_new:
            break
    return best
```

minimax with depth limit algorithm

In [5]:

```python
def minimax_depth(boardstate, depth, player):
    if player == COMP:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]

    if depth == 0 or game_over(boardstate):
        score = heauristic(boardstate)
        return [-1, -1, score]

    for cell in empty_cells(boardstate):
        x, y = cell[0], cell[1]
        boardstate[x][y] = player
        score = minimax_depth(boardstate, depth - 1, -player)
        boardstate[x][y] = 0
        score[0], score[1] = x, y
        # max value
        if player == COMP:
            if score[2] > best[2]:
                best = score
        # min value
        else:
            if score[2] < best[2]:
                best = score
    return best
```

minimax with depth limit and alpha beta pruning

In [6]:

```
def minimax_depth_alpha_beta(boardstate, depth, player, alpha, beta):
    alpha_new = alpha
    beta_new = beta
    if player == COMP:
        best = [-1, -1, -infinity, alpha, beta]
    else:
        best = [-1, -1, +infinity, alpha, beta]

    if depth == 0 or game_over(boardstate):
        score = heauristic(boardstate)
        return [-1, -1, score]

    for cell in empty_cells(boardstate):
        x, y = cell[0], cell[1]
        boardstate[x][y] = player
        score = minimax_depth_alpha_beta(boardstate, depth - 1, -player,alpha_new,beta_new)
        boardstate[x][y] = 0
        score[0], score[1] = x, y

        #max value
        if player == COMP:
            if score[2] > best[2]:
                best = score
            if score[2] > alpha_new:
                alpha_new = score[2]
        # min value
        else:
            if score[2] < best[2]:
                best = score
            if score[2] < beta_new:
                beta_new = score[2]
        #alpha beta
        if alpha_new >= beta_new:
            break
    return best
```

agent code for goal test, move validation and heuristics

In [7]:

```python
def eval_return(K,player):
    my_dict = {}
    main = [0 for i in range(K)]
    for i in range(1,K+1):
        main[0:i] = player*np.ones(i,dtype = int)
        l = list(permutations(main))
        l = list(set([i for i in l]))
        my_dict[10**(i-1)] = l
    return my_dict


def win_states(boardstate):
    boardstate = np.array(boardstate)
    win_state = []
    for i in start_end:
        for j in start_end:
            matrix = boardstate[i[0]:i[1]+1,j[0]:j[1]+1]
            for m in range(K):
                win_state.append(list(matrix[m,...]))
                win_state.append(list(matrix[...,m]))
            win_state.append(list(matrix.diagonal()))
            win_state.append(list(np.fliplr(matrix).diagonal()))
    return win_state

def heauristic(boardstate):
    win_state = win_states(boardstate)
    open_paths_comp = 0
    open_paths_human = 0

    for i in win_state:
        for j in my_dict_comp:
            if tuple(i) in my_dict_comp[j]:
                open_paths_comp+=j
        for j in my_dict_human:
            if tuple(i) in my_dict_human[j]:
                open_paths_human-=j
    score = open_paths_comp+open_paths_human
    return score

def wins(boardstate, player):
    win_state = win_states(boardstate)
```

```
    player_win = [player for i in range(K)]
    if player_win in win_state:
        return True
    else:
        return False

def evaluate(state):
    if wins(state, COMP):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else:
        score = 0

    return score

def game_over(state):
    return wins(state, HUMAN) or wins(state, COMP)


def empty_cells(state):
    cells = []

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


def set_move(x, y, player):
    if [x, y] in empty_cells(board):
        board[x][y] = player
        return True
    else:
        return False
```

ai agent moves deciding function

In [8]:

```
def ai_turn(c_choice, h_choice,algo_type):
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    print(f'Computer turn [{c_choice}]')
    draw_board(board, c_choice, h_choice)

    if algo_type == 5 and depth == N**2:
        choices = []
        choices.append(0)
        choices.append(N-1)

        x = choice(choices)
        y = choice(choices)

    elif depth == N**2:
        choices = []
        for i in range(0,N):
            choices.append(i)

        x = choice(choices)
        y = choice(choices)

    else:
        if(algo_type == 1):
            move = minimax(board,depth,COMP)
        elif(algo_type == 2):
            move = minimax_alpha_beta(board, depth, COMP, -infinity,+infinity)
        elif(algo_type == 3):
            if(depth >= 6):
                depth = 6
            move = minimax_depth(board,depth,COMP)
        elif algo_type == 4:
            if(depth >= 6):
                depth = 6
            move = minimax_depth_alpha_beta(board,depth, COMP, -infinity,+infinity)
#         move = minimax(board, depth, COMP)
        elif(algo_type == 5):
            if(depth >= 3):
                depth = 3
```

```
            move = minimax_depth_alpha_beta(board,depth, COMP, -infinity,+infinity)

        x, y = move[0], move[1]
#         print(x," x y ",y)

    can_move = set_move(x, y, COMP)
    if can_move:
        toc = mixer.Sound("hit.wav")
        toc.play()
#     print(board)
    time.sleep(1)
```

human turn moves maping to the board function

In [9]:

```
def human_turn(c_choice, h_choice):
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    print(f'Human turn [{h_choice}]')
    draw_board(board, c_choice, h_choice)

    can_move = False

    while(not can_move):
        for event in pygame.event.get():
            if event.type is pygame.MOUSEBUTTONDOWN and canPlay:
                (mouseX, mouseY) = pygame.mouse.get_pos()
                (column, row) = map_mouse_to_board(mouseX, mouseY)
                can_move = set_move(row,column,HUMAN)
                if can_move:
                    tick = mixer.Sound("toc.wav")
                    tick.play()
                break

        pygame.display.update()
        if can_move:
#             print(board)
            break
```

mapping mouse with pygame board and printing the board

In [10]:

```
def map_mouse_to_board(x, y):
    for i in range(0,board_size):
        if margin + (gameSize / board_size) * i <= x < margin + (gameSize / board_size) * (i + 1):
            column = i

    for i in range(0,board_size):
        if margin + (gameSize / board_size) * i <= y < margin + (gameSize / board_size) * (i + 1):
            row = i
    return column, row



def draw_lines():
    # vertical lines
    for i in range(0,board_size+1):
        pygame.draw.line(screen, lineColor, (margin + (gameSize // board_size) * i, margin),
                         (margin + (gameSize // board_size) * i, screenSize - margin), lineSize)
    # horizontal lines
        pygame.draw.line(screen, lineColor, (margin, margin + (gameSize // board_size) * i),
                         (screenSize - margin, margin + (gameSize // board_size) * i), lineSize)

def draw_board(state, c_choice, h_choice):

    chars = {
        -1: h_choice,
        1: c_choice,
        0: ' '
    }


    myFont = pygame.font.SysFont('Tahoma', gameSize // board_size)
    x = 0
    for row in state:
        y = 0
        for cell in row:
            if cell == h_choice:
                cell = -1
            elif cell == c_choice:
                cell = +1

            symbol = chars[cell]
```

```
            sentstring = ''
            color = ''
            if symbol == xMark:
                color = xColor
                sentstring = 'X'
            elif symbol == oMark:
                color = oColor
                sentstring = '0'
            else:
                color = oColor
                sentstring = ''

            text_surface = myFont.render(sentstring, False, color)
            screen.blit(text_surface, (y * (gameSize // board_size) + margin + (gameSize // (board_size
            pygame.display.update()
            y = y + 1
        pygame.display.update()
        x = x + 1
```

main function

In [11]:

```
def main(h_choice,first,algo_type):
    c_choice = ''

    canPlay = True

    # Setting computer's choice
    if h_choice == 'X':
        c_choice = 'O'
    else:
        c_choice = 'X'

    myFont = pygame.font.SysFont('Tahoma', 20)
    rect1 = pygame.Rect(margin-4,margin-15,360, 25 )
    rect2 = pygame.Rect(screenSize-410,margin-15,377, 25 )

    # pygame.quit()
    while True:
        mixer.music.load("background.wav")
        mixer.music.play(-1)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                print("QUIT")
                pygame.display.quit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_r:
                    screen.fill(backgroundColor)
                    draw_lines()
                    canPlay = True
                if event.key == pygame.K_ESCAPE:
                    print("ESCAPE")
                    pygame.display.quit()
#             print('hello')
            draw_board(board,c_choice,h_choice)
            pygame.display.update()
            while len(empty_cells(board)) > 0 and not game_over(board):
#                 print("in while")
                screen = pygame.display.get_surface()

                pygame.draw.rect(screen,(255,255,255),rect2)
                pygame.display.flip()
                text_surface = myFont.render(algorithm_choosen, True,(255,0,0))
```

```
                screen.blit(text_surface, (screenSize-360,margin-15))
                draw_board(board,c_choice,h_choice)
                pygame.display.update()

                if first == 'N':
                    start_time = time.time()
                    ai_turn(c_choice, h_choice,algo_type)
                    taken_time = "agent taken time: "+str(time.time() - start_time)
                    pygame.draw.rect(screen,(255,255,255),rect1)
                    pygame.display.flip()
                    print(taken_time)
                    text_surface = myFont.render(taken_time, True,(255,0,0))
                    screen.blit(text_surface, (margin,margin-15))
                    draw_board(board,c_choice,h_choice)
                    pygame.display.update()

                    first = ''

                human_turn(c_choice, h_choice)

                start_time = time.time()
                ai_turn(c_choice, h_choice,algo_type)
                taken_time = "agent taken time: "+str(time.time() - start_time)
                print(taken_time)
                pygame.draw.rect(screen,(255,255,255),rect1)
                pygame.display.flip()
                text_surface = myFont.render(taken_time, True,(255,0,0))
                screen.blit(text_surface, (margin,margin-15))
                draw_board(board,c_choice,h_choice)
                pygame.display.update()
            draw_board(board,c_choice,h_choice)



            pygame.display.update()
#                   winner = get_winner(board)
            if wins(board,HUMAN):
                myFont = pygame.font.SysFont('Tahoma', screenSize // 5)
                screen.fill(backgroundColor)
                text_surface = myFont.render(h_choice+" won!", False, (255,255,255))
                screen.blit(text_surface, (margin + screenSize // 10, screenSize // 2 - screenSize // 1
#                 print(" h won")
```

```
                mixer.music.load("win.ogg")
                mixer.music.play()
                pygame.display.update()
                canPlay = False
#                 pygame.quit()
                return
            elif wins(board,COMP):
                myFont = pygame.font.SysFont('Tahoma', screenSize // 5)
                screen.fill(backgroundColor)
                text_surface = myFont.render(c_choice+" won!", False, (255,255,255))
                screen.blit(text_surface, (margin + screenSize // 10, screenSize // 2 - screenSize // 1
                mixer.music.load("win.ogg")
                mixer.music.play()
                pygame.display.update()
                canPlay = False
#                 print("C won")
#                 pygame.quit()
                return
            elif len(empty_cells(board)) == 0:
                myFont = pygame.font.SysFont('Tahoma', screenSize // 5)
                screen.fill(backgroundColor)
                text_surface = myFont.render("Draw!", False, (255,255,255))
                screen.blit(text_surface, (margin + screenSize // 5, screenSize // 2 - screenSize // 10
                mixer.music.load("draw.ogg")
                mixer.music.play()
                pygame.display.update()
                canPlay = False
#                 print('draw')
#                 pygame.quit()
                return
        pygame.display.update()

    # Main loop of this game
    while len(empty_cells(board)) > 0 and not game_over(board):
        if first == 'N':
            ai_turn(c_choice, h_choice)
            first = ''

        human_turn(c_choice, h_choice)
        ai_turn(c_choice, h_choice)
```

tkinter library for taking inputs from user

In [12]:

```
class Input(tk.Frame):

    def __init__(self, parent):

        tk.Frame.__init__(self, parent)
        self.parent = parent

        choose = ["X","O"]

        self.choose_selection = tk.StringVar()
        self.choose_selection.set(choose[0])

        self.choose_label = tk.Label(root, text="Choose your coin to start : ",font=('arial',20))
        self.choose_entry = tk.OptionMenu(root, self.choose_selection, *choose)
        self.choose_entry.config(font=('arial',13,'bold'))

        self.choose_label.grid(row=0, column=0, padx=80, pady=(430,0))


        self.choose_entry.grid(row=0, column=1, pady=(430,0))

        size = ["3","4","5","6","7"]

        self.size_selection = tk.StringVar()
        self.size_selection.set(size[0])

        self.size_label = tk.Label(root, text="Choose Board size : ",font=('arial',20))
        self.size_entry = tk.OptionMenu(root, self.size_selection, *size)
        self.size_entry.config(font=('arial',13,'bold'))

        self.size_label.grid(row=1, column=0, padx=5, pady=5)


        self.size_entry.grid(row=1, column=1, pady=5)

        win_size = ["3","4","5","6","7"]

        self.win_size_selection = tk.StringVar()
        self.win_size_selection.set(win_size[0])

        self.win_size_label = tk.Label(root, text="Number of coins in row to win : ",font=('arial',20))
```

```
        self.win_size_entry = tk.OptionMenu(root, self.win_size_selection, *win_size)
        self.win_size_entry.config(font=('arial',13,'bold'))

        self.win_size_label.grid(row=2, column=0, padx=5, pady=5)


        self.win_size_entry.grid(row=2, column=1, pady=5)

        start = ["Y","N"]

        self.start_selection = tk.StringVar()
        self.start_selection.set(start[0])

        self.start_label = tk.Label(root, text="Do you want to start the game? ",font=('arial',20))
        self.start_entry = tk.OptionMenu(root, self.start_selection, *start)
        self.start_entry.config(font=('arial',13,'bold'))

        self.start_label.grid(row=3, column=0, padx=5, pady=5)

        self.start_entry.grid(row=3, column=1, pady=5)

        algorithms = ["1 Normal min-max","2 Alpha-Beta","3 Depth_Limit","4 Depth_limit+alpha_beta","5 s

        self.algo_selection = tk.StringVar()
        self.algo_selection.set(algorithms[0])

        self.algo_label = tk.Label(root, text="To which algorithm you want to play? ",font=('arial',20)
        self.algo_entry = tk.OptionMenu(root, self.algo_selection, *algorithms)
        self.algo_entry.config(font=('arial',13,'bold'))

        self.submit_button = tk.Button(text="Submit",font=('arial',20), command=self.close_window)

        self.algo_label.grid(row=4, column=0, padx=5, pady=5)
        self.submit_button.grid(columnspan=2, row=5, column=0, padx=50, pady=5)

        self.algo_entry.grid(row=4, column=1, pady=5)

        self.warning1 = tk.Label(root, text="Any algorithm will take not more than 6sec for a 3x3 matri
        self.warning1.grid(row=6,column=0, padx=5, pady=5)
        self.warning2 = tk.Label(root, text="1 & 2 algorithms will take more time for a 4x4 and above b
        self.warning2.grid(row=7,column=0, padx=5, pady=5)
        self.warning3 = tk.Label(root, text="check the readme file for the stats for a 4x4 and above bo
```

```
        self.warning3.grid(row=8,column=0, padx=5, pady=5)

        # styling
        bglabelcolor = (206,234,230)
        self.warning1.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (255,0,0))
        self.warning2.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (255,0,0))
        self.warning3.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (255,0,0))
        self.choose_label.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (41,27,79))
        self.size_label.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (41,27,79))
        self.start_label.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (41,27,79))
        self.algo_label.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (41,27,79))
        self.win_size_label.configure(bg='#%02x%02x%02x' % bglabelcolor,fg='#%02x%02x%02x' % (41,27,79)
        self.choose_entry.configure(bg='#%02x%02x%02x' % (3,0,99),fg='#%02x%02x%02x' % (216,141,25))
        self.size_entry.configure(bg='#%02x%02x%02x' % (3,0,99),fg='#%02x%02x%02x' % (253,213,44))
        self.start_entry.configure(bg='#%02x%02x%02x' % (3,0,99),fg='#%02x%02x%02x' % (253,213,44))
        self.algo_entry.configure(bg='#%02x%02x%02x' % (3,0,99),fg='#%02x%02x%02x' % (253,213,44))
        self.win_size_entry.configure(bg='#%02x%02x%02x' % (3,0,99),fg='#%02x%02x%02x' % (253,213,44))
        self.submit_button.configure(bg='#%02x%02x%02x' % (3,0,99),fg='#%02x%02x%02x' % (253,213,44))




    def close_window(self):
        self.choose_type = self.choose_selection.get()
        self.start_type = self.start_selection.get()
        self.algo_type = self.algo_selection.get()
        self.board_size = self.size_selection.get()
        self.win_size = self.win_size_selection.get()
        self.quit()
```

global variables and to start the game

In [13]:

```
# tkinter window for taking inputs from user

root = tk.Tk()
root.geometry("1200x820")
root.title("Tic Tac Toe")
img  = Image.open("tictactoe.gif")
img  = img.resize((1200,400), Image.ANTIALIAS)
photo=ImageTk.PhotoImage(img)
lab = tk.Label(image=photo).place(x=0,y=0)
root.configure(bg='#%02x%02x%02x' % (206,234,230))
app = Input(root)
root.mainloop()

h_choice = app.choose_type
first = app.start_type
algo_type = app.algo_type
algorithm_choosen = str(algo_type)
board_size = int(app.board_size)
win_size = int(app.win_size)
algo_type = int(algo_type[0])
root.destroy()

#-----globally declared variables-------

HUMAN = -1
COMP = +1
diff = 0
board = []
start_end = []
my_dict_comp = {}
my_dict_human = {}
start_end = []
K = win_size
N = board_size
diff = N-K+1
list_f = [i for i in range(N)]
for i in range(diff):
    start_end.append([list_f[0+i],list_f[K+i-1]])
my_dict_comp = eval_return(K,COMP)
my_dict_human = eval_return(K,HUMAN)
board = np.zeros((N,N),dtype = int).tolist()
```

```
42
43
44 # ------start the main function and pygame window------
45 screenSize = 800
46 margin = 40
47 gameSize = 800 - (2 * margin)
48 lineSize = 10
49 backgroundImage = pygame.image.load("bg.jpeg")
50 backgroundColor = (0,0,0)
51 lineColor = (255,255,255)
52 xColor = (200, 0, 0)
53 oColor = (0, 0, 200)
54 xMark = 'X'
55 oMark = 'O'
56 pygame.display.init()
57 pygame.mixer.init()
58 screen = pygame.display.set_mode((screenSize, screenSize))
59 pygame.display.set_caption("Tic Tac Toe")
60 pygame.font.init()
61 myFont = pygame.font.SysFont('Tahoma', gameSize // board_size)
62 # screen.fill(backgroundColor)
63 screen.blit(backgroundImage,[0,0])
64 canPlay = True
65 # print("draw_lines()")
66 draw_lines()
67 main(h_choice,first,algo_type)
68 time.sleep(2)
69 pygame.mixer.quit()
70 pygame.display.quit()
```

```
Human turn [X]
Computer turn [O]
agent taken time: 1.2924182415008545
Human turn [X]
Computer turn [O]
agent taken time: 1.0464446544647217
Human turn [X]
Computer turn [O]
agent taken time: 1.0344634056091309
Human turn [X]
Computer turn [O]
agent taken time: 1.0351033210754395
```

```
Human turn [X]
agent taken time: 1.7642974853515625e-05
```

In [ ]:

```
1
```