دانشگاه صنعتي اميركبير
(پلی تکنیک تهران)
دانشکده علوم کامپیوتر

# گزارش 3.2

نگارش
محمدرضا اردستانی

استاد راهنما
مهدی قطعی

فروردین 1399

# فـهـرست مـطـالـب

صفـحه

# فـصل اول
# adversarial search

# مقدمه

کلاس بزرگی از مسئله های بشری در رده ی سرچ در درخت ( یا گراف) قرار میگیرد. بازی ها نیز اکثرا قابل تبدیل شدن به قالب مسئله ی جستجو در درخت یا گراف هستند پس میتوان آن ها را با کمک این قبیل الگوریتمها (سرچ) حل نمود. بازی ها (خیلی از روابط انسان ها هم شبیه بازی ها عمل میکند) نیز رده های مختلفی دارند. یکی از رده های آن این است که دو ( یا چند) بازیکن هستند که هر دو به دنبال کم تر کردن سود دیگری برای به دست آوردن سود بیشتر هستند، زیرا که منابع محدود است. البته همه بازی ها به این شکل نیستند. اسم این رده ی بازی ها **adversarial** میگوند.

در ادامه با یکی از روشهای مخصوص حل این گونه مسائل ( **minimax algorithm and alpha-beta pruning**) به حل مسئله **tic-tac-toe** میپردازیم.

# فـصل د و م
# tic tac toe using Minimax and Alpha-beta pruning

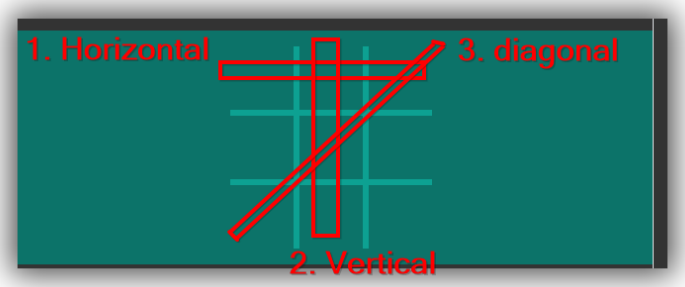
## 2-1- توضیحات الگوریتم و کد برنامه

این برنامه با پایتون و کتاب خانه معروف ساخت گیم با پایتون به نام pygame ایمپلیمنت شده است. از کتابخانه pygame براس ساخت برد بازی استفاده شده است که با کلیک بر روی خانه های آن انتخاب شما ذخیره میشود. برای بخش های مختلف برنامه از سایت ها و ویدئوهای آموزشی مختلفی استفاده شده است که همگی در قسمت منابع و بخش استفاده شدن آن ها آورده شده اند.

در انتهای فایل، بخش پیوست، کد برنامه قابل دسترس هست.

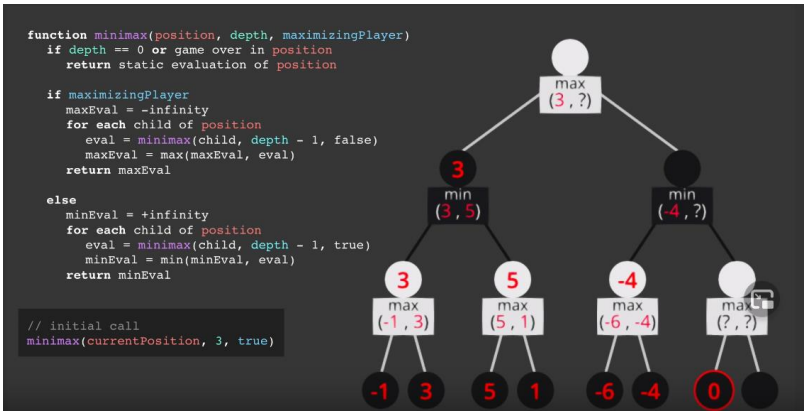
***توضیحات مربوط به خود بازی:***

بازی tic-tac-toe بازی کلاسیک ساده ای است که final state های آن برنده شدن یک بازیکن و یا مساوی tie شدن بازی می باشد. حالتهای بد در بازی به صورت زیر است:



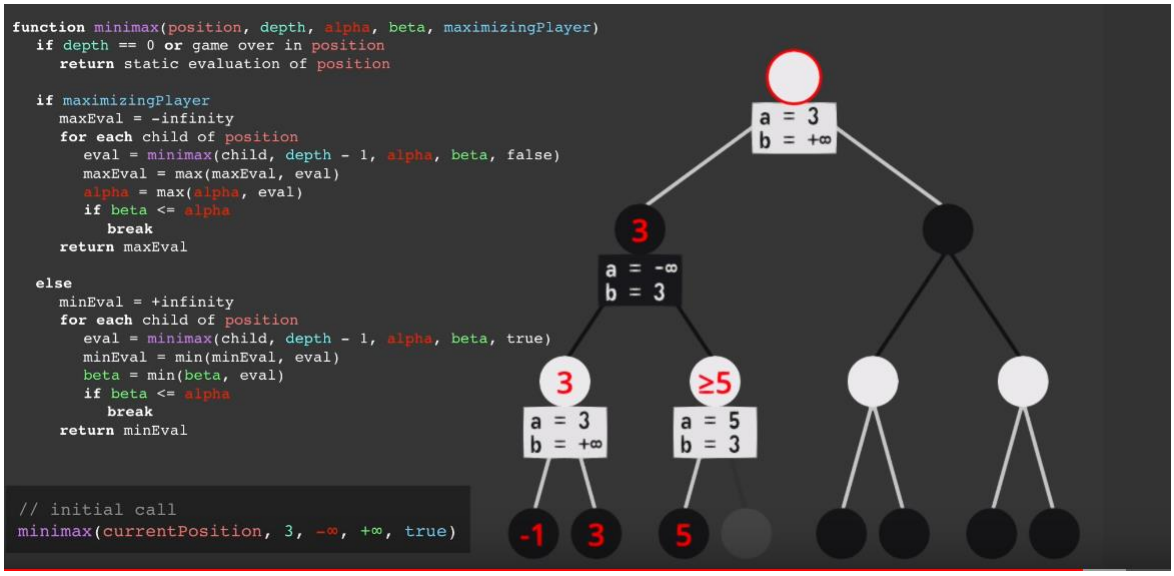

سه حالت کلی قطری، عمودی و افقی وجود دارد که منجر به برد میشود.تساوی، پر شدن تمام خانه های جدول میباشد.

***پیاده سازی Minimax :***



الگوریتم مینمکس با توجه با تصویر بالا به صورت recursive اجرا میشود ( آدرس منبع بالا در انتها آمده است)

***پیاده سازی alpha-beta pruning :***

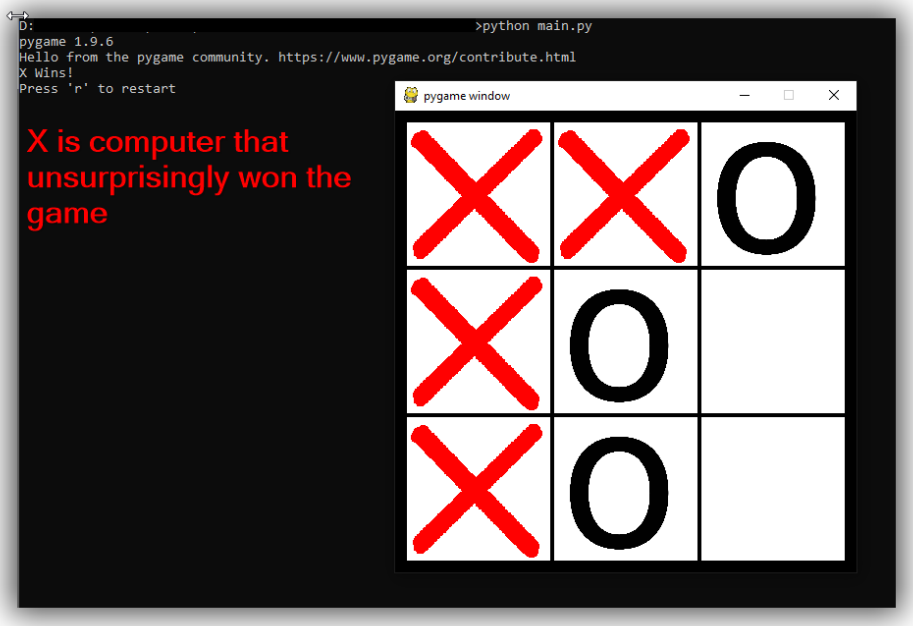


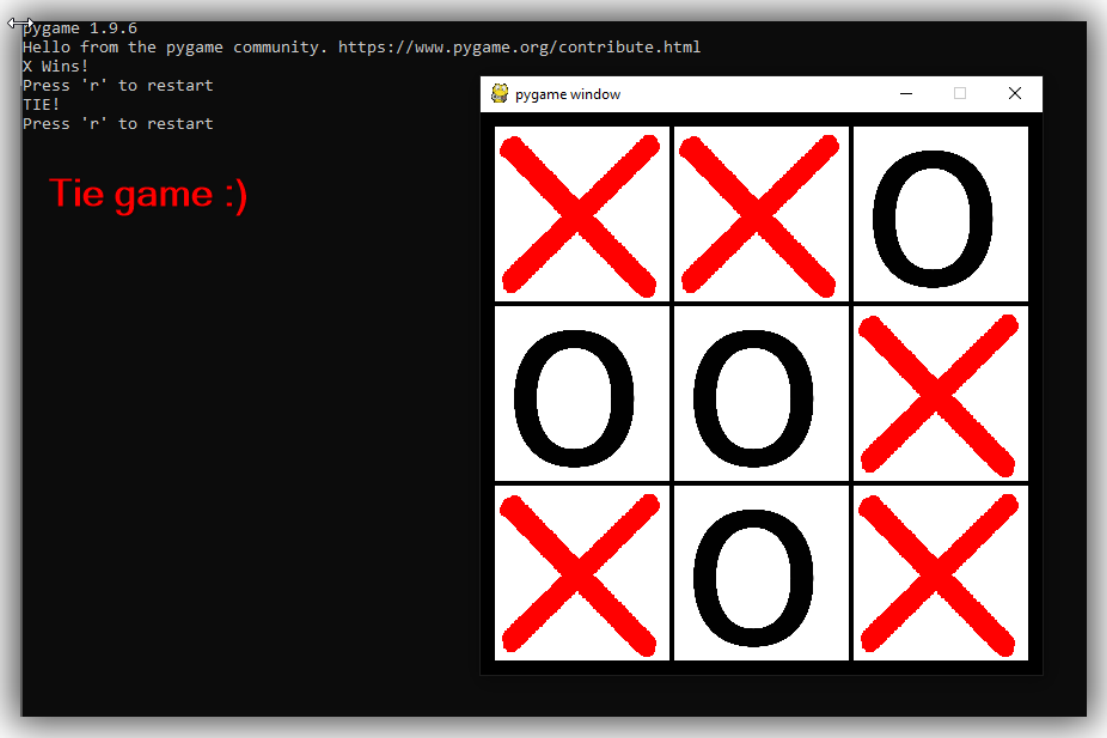

همین طور برای پیاده سازی alpha-beta pruning از تصویر بالا الهام گرفته شده است.

**بخش خلاقانه برای پیاده سازی بازی(استفاده از pygame) :**

برای داشتن یک UI مناسب از کتاب خانه ی معروف پایتون برای بازی ها pygame استفاده شده است.

## 2-2- نمونه ای از اجرای برنامه



# منابع و مراجع

برای پیاده سازی از این سایت الهام گرفته ام:


[for using pygame library]
(https://www.youtube.com/watch?v=ujOTNg17LjI&list=PLQVvvaa0QuDdLkP8MrOXLerKuf6r80KO)
(https://www.pygame.org/docs/)

[Minimaz algorithm and alpha-beta pruning]
(https://www.youtube.com/watch?v=STjW3eH0Cik&t=9s)
(https://www.youtube.com/watch?v=trKjYdBASyQ&t=348s)
(https://www.youtube.com/watch?v=l-hh51ncgDI)

Appendix :

کد برنامه به زبان پایتون :

```
import pygame
#links for getting introduced to pygame
#https://www.youtube.com/watch?v=ujOTNg17LjI&list=PLQVvvaa0QuDdLkP8MrOXLe_rKuf6r80K
O
#http://help.codakid.com/en/articles/2551585-how-to-install-pygame
pygame.init()
import math
import time

#defining pygame GUI objects and functions
SIZE = WIDTH, HEIGHT = 470, 470

def rect(screen, color, x, y, w, h, fill=0):
    pygame.draw.rect(screen, color, (x, y, w, h), fill)

def square(screen, color, x, y, s, fill=0):
    rect(screen, color, x, y, s, s, fill)

def ellipse(screen, color, x, y, w, h, fill=0):
    pygame.draw.ellipse(screen, color, (x, y, w, h), fill)

def circle(screen, color, x, y, r, fill=0):
    ellipse(screen, color, x, y, r, r, fill)

def background(screen, color):
    rect(screen, color, 0, 0, WIDTH, HEIGHT)
# end of pygame initialization basic properties and creating its functions

# We have there general situation for winning: Diagonal, Horizental, vertical
def checkWinner(board):
    n = len(board)
    first = board[0][0]

    #cheking diagonal
    diagonal = first != ""
    for i in range(n):
        if board[i][i] != first:
            diagonal = False
            break
    if diagonal:
        return first
    first = board[0][n-1]
    back_diag = first != ""
    for i in range(1, n+1):
```

```
        if board[i-1][n-i] != first:
            back_diag = False
            break
    if back_diag:
        return first
    #cheking Horizental and vertical
    for i in range(n):
        first = board[i][0]
        sideways = first != ""
        for j in range(n):
            if board[i][j] != first:
                sideways = False
        if sideways:
            return first

    for i in range(n):
        first = board[0][i]
        # print(first)
        sideways = first != ""
        for j in range(n):
            if board[j][i] != first:
                sideways = False
        if sideways:
            return first
    #Check if it's tie game
    open_spots = 0
    for i in range(n):
        for j in range(n):
            if board[i][j] == "":
                open_spots += 1
    if open_spots == 0:
        return "tie"
    return None
# end of winnerCheck function


def best_move(board):
    n = len(board)
    best_score = -math.inf
    move = (0, 0)
    for i in range(n):
        for j in range(n):
            if board[i][j] == "":
                board[i][j] = "x"
                score = minimax(board, 0, False, len(board))
                board[i][j] = ""
                if score > best_score:
                    best_score = score
                    move = (i, j)
```

```
    board[move[0]][move[1]] = "x"
    return board

scores = {"x" : 10, "o" : -10, "tie": 0}

def minimax(board, depth, is_max, n, alpha = -math.inf, beta = math.inf):
    winner = checkWinner(board)
    if winner:
        # print(depth)
        return scores[winner]
    if is_max:
        best_score = -math.inf
        for i in range(n):
            for j in range(n):
                if board[i][j] == "":
                    board[i][j] = "x"
                    score = minimax(board, depth+1, False, n, alpha, beta)
                    board[i][j] = ""
                    best_score = max(score, best_score)
                    alpha = max(alpha, score)
                    if beta >= alpha:
                        pass
        return best_score
    else:
        best_score = math.inf
        for i in range(n):
            for j in range(n):
                if board[i][j] == "":
                    board[i][j] = "o"
                    score = minimax(board, depth+1, True,n, alpha, beta)
                    board[i][j] = ""
                    best_score = min(score, best_score)
                    beta = min(beta, score)
                    if alpha >= beta:
                        pass
        return best_score

def reset(n):
    board = [["" for i in range(n)] for j in range(n)]
    loop = True
    return board, loop, None, True

#Driver code
def main():
    padding = 10
    n = 3
    s = (WIDTH-padding*2)//n

    board = [["" for i in range(n)] for j in range(n)]
```

```
    turn = "x"

    x_image = pygame.image.load(r"ex.png")
    x_image = pygame.transform.scale(x_image, (s, s))
    o_image = pygame.image.load(r"o.png")
    o_image = pygame.transform.scale(o_image, (s, s))

    loop = True
    gameover = False

    frame_count = 0
    human_played = False
    winner = False
    restarted = False

    running = True
    board = best_move(board)
    turn = "x" if turn == "o" else "o"
    # print(f"{turn}'s Turn")

    screen = pygame.display.set_mode(SIZE) #Start the screen

    while running:
        prev = frame_count
        Mouse_x, Mouse_y = pygame.mouse.get_pos()
        for event in pygame.event.get():
                if event.type == pygame.QUIT: #The user closed the window!
                    running = False #Stop running
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_r:
                        board, loop, winner, restarted = reset(n)

                if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
                    j = int(Mouse_x//s)
                    i = int(Mouse_y//s)
                    if i < n and j < n:
                        if board[i][j] == "":
                            board[i][j] = turn
                            turn = "x" if turn == "o" else "o"
                            human_played = True
                        winner = checkWinner(board)

        if loop:
            rect(screen, (255, 255, 255), padding, padding, WIDTH-
padding*2, HEIGHT-padding*2)

            # Logic goes here
            if not winner:
```

```
                winner = checkWinner(board)
            # print(winner)
            if winner:
                if winner == "tie":
                    print(winner.upper()+"!")
                else:
                    print(winner.upper(), "Wins!")
                print("Press 'r' to restart")
                loop = False

            for i in range(n):
                for j in range(n):
                    item = board[i][j]
                    if item == "x":
                        screen.blit(x_image, (j*s+padding, i*s+padding))
                    elif item == "o":
                        screen.blit(o_image, (j*s+padding, i*s+padding))
                    square(screen, (0,0, 0), j*s+padding, i*s+padding, s, 3)


            pygame.display.update()

            if restarted:
                turn = "x"
                board = best_move(board)
                turn = "x" if turn == "o" else "o"
                restarted = False

            if human_played:
                time.sleep(.5)
                board = best_move(board)
                turn = "x" if turn == "o" else "o"
                # print(f"{turn}'s Turn")
                human_played = False
            frame_count += 1
    pygame.quit() #Close the window

if __name__ == "__main__":
    main()
```