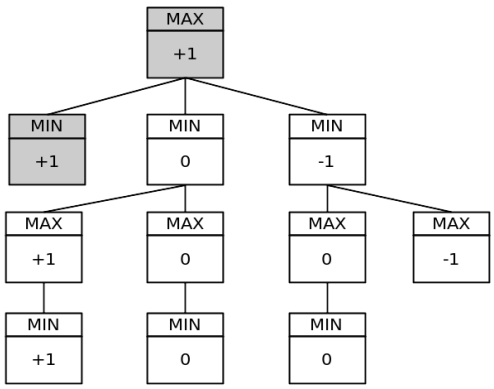
به نام خدا

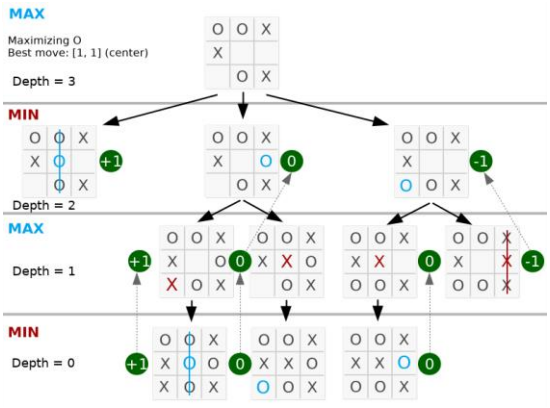
حدیث غفوری

شماره دانشجویی: 9825413

برای پیاده سازی الگوریتم minimax از الگوریتم زیر ایده گرفته شده است.

```
minimax(state, depth, player)

        if (player = max) then
                best = [null, -infinity]
        else
                best = [null, +infinity]

        if (depth = 0 or gameover) then
                score = evaluate this state for player
                return [null, score]

        for each valid move m for player in state s do
                execute move m on s
                [move, score] = minimax(s, depth - 1, -player)
                undo move m on s

                if (player = max) then
                        if score > best.score then best = [move, score]
                else
                        if score < best.score then best = [move, score]

        return best
end
```





پیاده سازی:

```python
def minimax(board, player, depth):
    if player == player1:
        best_value = -inf
    else:
        best_value = +inf

    if depth == 0 or is_game_over(board):
        score = evaluate(board)
        return None, score

    global num_repeat_algorithm
    num_repeat_algorithm += 1

    score = None      You, 1 hour ago • add first version of hw2 AI
    best_move = None

    for i in range(BOARD_SIZE):
        if board[i] == ' ':
            board[i] = player
            _, score = minimax(board,
                               G_player if player == R_player else R_player,
                               depth - 1)
            board[i] = ' '
            if player == player1:  # maximaize-player1
                if score > best_value:
                    best_value = score
                    best_move = i
            else:
                if score < best_value:  # minimize-player2
                    best_value = score
                    best_move = i

    return best_move, best_value
```

فرض شده است که بازیکن اول یا همان player1 بازیکن max است و بازیکن دوم بازیکن min است که میخاهد سود را کمینه کند.

اگر به عمق صفر برسیم یا بازی به جواب نهایی برسید و گیم اور اعلام شود، یعنی در برگ های درخت قرار داریم و باید مقدار ترمینال ها محاسبه شود برای اینکار از یک تابع ارزیابی استفاده شده.

در حلقه ای به تعداد تکرار ۲۵ (چون برد بازی ۵*۵ است) شروع به ساخت درخت میکنیم. ما فقط باید حالاتی از برد را درنظر بگیریم که خانه خالی باشد پس چک میکنیم آن خانه خالی است یا نه.

به طور موقتی مقدار مهره بازیکن را در ان خانه قرار میدهیم و بهترین مقداری که از فرزندان ان حالت ایجاد میشود را بررسی میکنیم یعنی دوباره باید تابع را صدا کنیم ولی این بار برای حریف بازیکن فعلی پس بررسی میکنیم اگر الان بازیکن فعلی مهره ی سبز را دارد بازیکن با مهره ی قرمز را فراخوانی کنیم.

تابع minimax دو مقدار برمیگرداند. یکی بهترین اکشن و دیگری بهترین امتیاز برای ان بازیکن

اینجا نیاز به بهترین امتیاز داریم پس برای هر بازیکن بسته به اینکه ماکس است یا مین،باید بهترین امتیازش را اپدیت کنیم و بهترین اکشن هم اندیس حلقه درنظر میگیریم که همان i است.

در نهایت این دو مقدار برگردانده میشوند تا برای نودهای بالاتر استفاده شوند.

```
def evaluate(board):
    """
    Function to heuristic evaluation of state.
    return: +1 if the player1 wins; -1 if the player2 wins; 0 draw
    """
    winner = check_win(board)

    if winner == player1:
        score = +1
    elif winner == player2:
        score = -1
    else:
        score = 0

    return score
```

چون بازیکن اول ماکس است، امتیاز 1 میگیرد و بازیکن دوم که مین است، 1- و در غیر این صورت صفر داده میشود.

تابع بررسی برنده

```
248 def check_win(board):
249     """
250     This function checks for all possible three-in-a-row wins on a
251     5x5 tic-tac-toe board.
252     """
253     # Possible Horizontal Wins
254     horizontal = [0, 1, 2, 5, 6, 7, 10, 11, 12, 15, 16, 17, 20, 21, 22]
255     for i in horizontal:
256         if (board[i] != ' '
257             and board[i] == board[i + 1] == board[i + 2]):
258             return board[i]
259
260     # Possible Vertical wins
261     for i in range(15):
262         if (board[i] != ' '
263             and board[i] == board[i + 5] == board[i + 10]):
264             return board[i]
265
266     # Possible Diagonal wins, left to right
267     l_diagonal = [0, 1, 2, 5, 6, 7, 10, 11, 12]
268     for i in l_diagonal:
269         if (board[i] != ' '
270             and board[i] == board[i + 5 + 1] == board[i + 10 + 2]):
271             return board[i]
272
273     # Possible Diagonal wins, right to left
274     r_diagonal = [2, 3, 4, 7, 8, 9, 12, 13, 14]
275     for i in r_diagonal:
276         if (board[i] != ' '
277             and board[i] == board[i + 5 - 1] == board[i + 10 - 2]):
278             return board[i]
```

تابع حالت های مختلف در درخت را بررسی میکند تا اگر بازی به جوابی برسد، مشخص شود.

حالت هایی که در آن به یک جواب برای بازی دست پیدا میکنیم به شرح زیر است:

- در آن یکی از بازیکن ها برنده میشوند.
  - بازیکن برنده، ۳ مهره ی خود را در ۳ خانه پشت سرهم در یک سطر قرارداده است.
  - بازیکن برنده، ۳ مهره ی خود را در ۳ خانه پشت سرهم در یک ستون قرارداده است.
  - بازیکن برنده، ۳ مهره ی خود را در ۳ خانه پشت سرهم در قطر اصلی (از چپ به راست) قرارداده است.
  - بازیکن برنده، ۳ مهره ی خود را در ۳ خانه پشت سرهم در قطر فرعی (از راست به چپ) قرارداده است.
- هر دو بازیکن مساوی کنند.

تابع زیر بررسی میکند که آیا بازی به انتها رسیده است یا خیر.

```python
def is_game_over(board):
    winner = check_win(board)
    return winner == G_player or winner == R_player

```

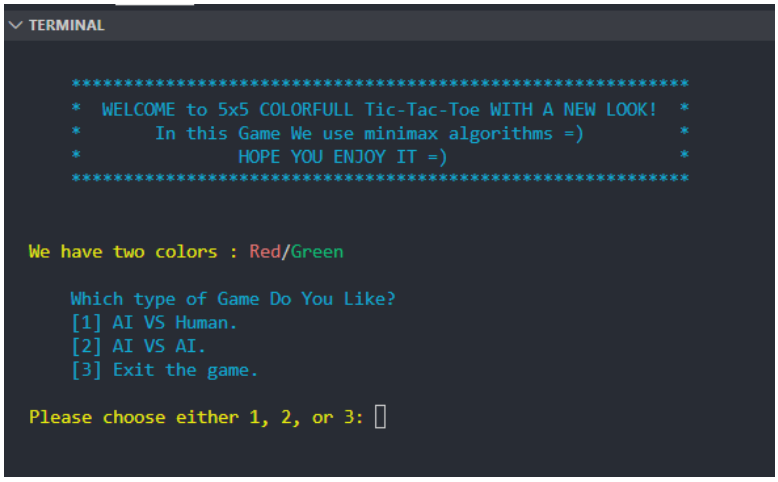
الگوریتم alpha beta minimax

```python
def minimax_ab(board, player, depth, alpha=-inf, beta=inf):
    if depth == 0 or is_game_over(board):
        score = evaluate(board)
        return None, score

    global num_repeat_algorithm
    num_repeat_algorithm += 1

    best_move = None
    value = None

    for i in range(BOARD_SIZE):
        if board[i] == ' ':
            board[i] = player
            _, value = minimax_ab(board,
                                  G_player if player == R_player else R_player,
                                  depth - 1,
                                  alpha,
                                  beta)
            board[i] = ' '
            if (value is not None):
                if player == player1 and value > alpha:
                    alpha = value
                elif player == player2 and value < beta:
                    beta = value
                best_move = i

                if (alpha != -inf and beta != inf
                        and alpha >= beta if player == player1 else beta >= alpha):
                    break

    if player == player1:
        return best_move, alpha

    return best_move, beta
```

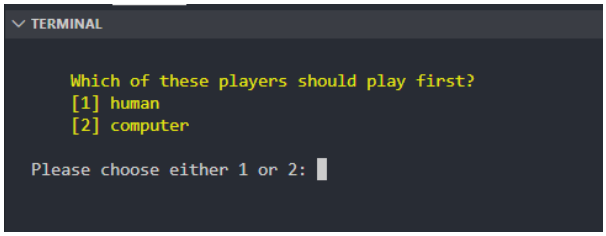
تفاوت این الگوریتم با مینیمکس، در به کارگیری یک الفا و بتا است که برای بازکین ماکس از آلفا و برای بازیکن مین از بتا استفاده میشود.

مراحل پیاده سازی کد به همراه خروجی

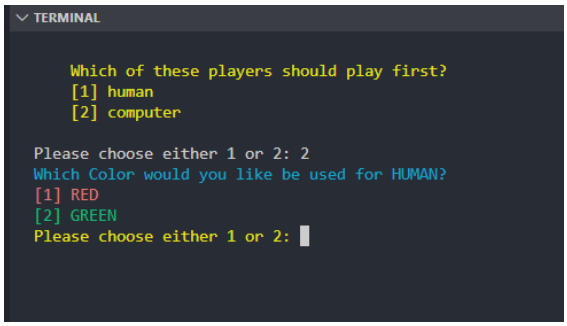
انتخاب نوع بازی



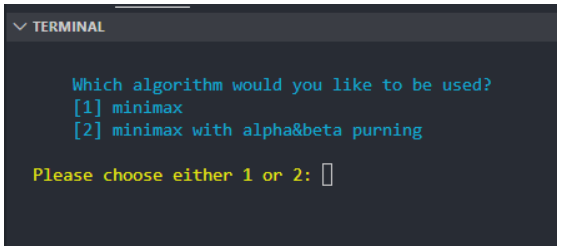
انتخاب بازیکن اول



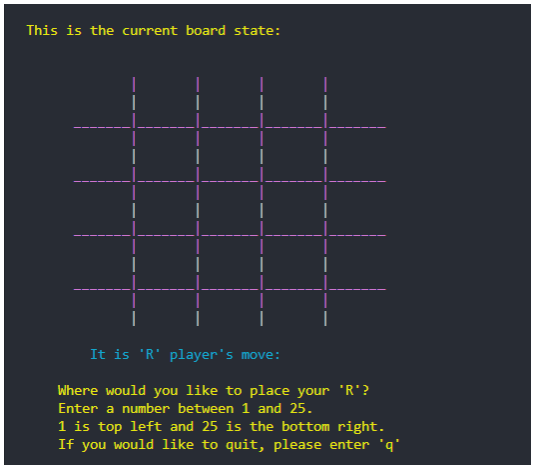
انتخاب رنگ یا مهره ی انتخابی



انتخاب الگوریتم بازی

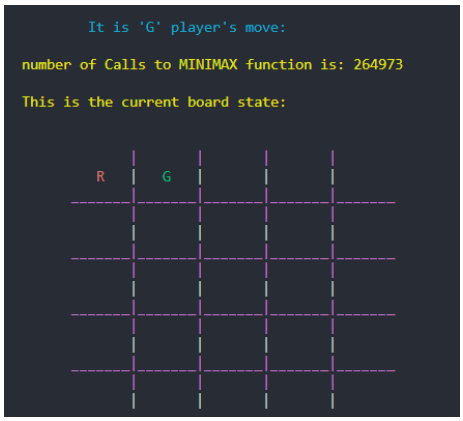


نمایش اولیه از برد بازی



نمایش بازیکنی که باید تصمیم گیری کند یا در حال تصمیم گیری است.

نمایش تعداد فراخوانی های الگوریتم.



نمایش برنده ی بازی

در پایان کاربر میتواند انتخاب کند که دوباره بازی کند یا نه.

تابع اصلی

```
def main():       You, 2 hours ago • add first version of hw2 AI
    show_welcome_messages()
    while True:
        board = board_game_init()
        selected_game_type = select_game_type()
        clean()
        if selected_game_type == QUIT_GAME:
            break

        elif selected_game_type == HUMAN_VS_AI_GAME:  # HUMAN VS AI
            select_first_player()
            player1 = select_player_color(HUMAN_PLAYER)

        elif selected_game_type == AI_VS_AI_GAME:  # AI VS AI
            player1 = select_player_color(AI_PLAYER)

        player2 = get_player2(player1)
        clean()
        selected_game_algorithm = select_game_algorithm()
        clean()
        game(board, player1, player2, selected_game_type, selected_game_algorithm)
        another_round = play_agian()
        if another_round:
            clean()
            continue
        else:
            break

    clean()
    print(colored("\n\nGoodbye.\n\n", 'magenta'))
```

در این تابع مراحل زیر اجرا میشود.

نمایش اطلاعات اولیه بازی و خوش امدگویی

```
def show_welcome_messages():
    welcome_message_string = """
    ************************************************************
    *  WELCOME to 5x5 COLORFULL Tic-Tac-Toe WITH A NEW LOOK!   *
    *       In this Game We use minimax algorithms =)          *
    *              HOPE YOU ENJOY IT =)                        *
    ************************************************************

    """
    print(colored(welcome_message_string, MESSAGES_COLOR2))
    print(
        f"{colored('We have two colors :',MESSAGES_COLOR1)} {colored('Red','red')}/{colored('Green','green')}")
```

در حلقه ی اصلی بازی که تا زمانی که دکمه ی q زده نشود از ان خارج نمی‌شویم،باید اطلاعات زیر را مشخص کنیم:

- نوع بازی (۱.بازی انسان در برابر کامپیوتر ۲. بازی کامپیوتر در برابر کامپیوتر)

- در صورت انتخاب نوع بازی ۱، باید تعیین شود که بازیکن اول انسان باشد یا کامپیوتر
- هر بازیکن، چه مهره یا رنگی را انتخاب میکند (از بین سبز و قرمز)
- انتخاب نوع الگوریتم برای اجرای بازی (۱. Minimax ۲. minimax با استفاده از هرس آلفا بتا)

با داشتن این اطلاعات، وارد تابع game میشویم که در انجا، بازی انجام میشود.

در انتهای این تابع main هم، از کاربر پرسیده میشود که آیا تمایل به اجرای مجدد بازی را دارد یا نه. اگر داشته باشد مجدد میتواند همه ی اطلاعات بالا را وارد کند و دوباره بازی کند در غیر این صورت از بازی خارج میشود.

انتخاب نوع بازی

```
def select_game_type():
    game_type_string = """
    Which type of Game Do You Like?
    [1] AI VS Human.
    [2] AI VS AI.
    [3] Exit the game.
    """
    print(colored(game_type_string, MESSAGES_COLOR2))
    selected_game_type = int(
        input(colored("Please choose either 1, 2, or 3: ", MESSAGES_COLOR1)))
    while selected_game_type not in (1, 2, 3):
        selected_game_type = int(
            input(colored("Please choose either 1, 2, or 3: ", MESSAGES_COLOR1)))

    return get_game_type(selected_game_type)
```

در تابع زیر، بازیکن دوم به کمک اطلاعاتی که از ورودی گرفتیم به دست می آید.

اگر بازیکن اول مهره ی قرمز را داشته باشد، بازیکن دوم مهره‌ی سبز میگیرد و اگر بازیکن اول مهره ی سبز داشته باشد، بازیکن دوم مهره‌ی قرمز میگیرد.

```
def get_player2(player1):
    if player1 == R_player:
        return G_player
    return R_player
```

کانفیگ اولیه ی board بازی

```
def board_game_init():
    global board
    board = []
    for _ in range(BOARD_SIZE):
        board.append(' ')
    return board
```

انتخاب مهره ی سبز یا قرمز

```
def select_player_color(player_type):
    print(colored('Which Color would you like be used for {}?'.format(
        player_type), MESSAGES_COLOR2))
    print(colored("[1] RED", 'red'))
    print(colored("[2] GREEN", 'green'))
    selected_color = int(
        input(colored("Please choose either 1 or 2: ", MESSAGES_COLOR1)))
    if selected_color not in (1, 2):
        print("Please choose again.")
        return
    return R_player if selected_color == 1 else G_player
```

۳. اگر نوع بازی، بازی انسان در برابر کامپیوتر باشد، از کاربر میپرسیم که کدام یک از این دو باید بازی را شروع کنند. انسان یا کامپیوتر؟

انتخاب الگوریتم بازی

```
def select_game_algorithm():
    select_algorithm_string = """
    Which algorithm would you like to be used?
    [1] minimax
    [2] minimax with alpha&beta purning
    """
    print(colored(select_algorithm_string, MESSAGES_COLOR2))
    selected_algorithm = int(
        input(colored("Please choose either 1 or 2: ", MESSAGES_COLOR1)))
    if selected_algorithm not in (1, 2):
        print("Please choose again.")
        return
    return MINIMAX_ALGORITHM if selected_algorithm == 1 else MINIMAX_AB_ALGORITHM
```

تابع حرکت انسان

در این تابع، حرکت دلخواه کاربر را از ورودی میگیریم و بررسی میکنیم که این حرکت معتبر باشد یعنی تابع is_player_move_valid فراخوانی میشود و تا زمانی که کاربر عدد مناسب وارد نکند، پیام وارد کردن عدد مناسب به اون نشان داده میشود.

عدد مناسب از ۱ تا ۲۵ است.

کاربر اگر در ورودی، کاراکتر 'q' را وارد کند، از بازی خارج میشود.

```python
def human_move(player_turn):
    move_message_str = f"""
    Where would you like to place your \'{player_turn}\'?
    Enter a number between 1 and 25.
    1 is top left and 25 is the bottom right.
    If you would like to quit, please enter \'q\'
    """
    print(colored(move_message_str, MESSAGES_COLOR1))
    move = -1
    while move == -1 or is_player_move_valid(board, move) == False:
        try:
            move = input(colored("Please enter your move: ", MESSAGES_COLOR2))
            if move == "q":
                return "q"
            move = int(move) - 1
            if not is_player_move_valid(board, move):
                move = -1
            else:
                board[move] = player_turn
                return
        except (KeyError, ValueError):
            print('enter a valid move.')
        except (EOFError, KeyboardInterrupt):
            print('Bye')      You, 19 minutes ago • Uncommitted changes
            exit()
```

تابع زیر برای ایجاد اطمینان از معتبر بودن ورودی کاربر، فراخوانی میشود.

```python
def is_player_move_valid(board, move):
    return 0 <= move <= 24 and board[move] not in (G_player, R_player)
```

حرکت کامپیوتر

در این تابع، کامپیوتر (Ai) بسته به نوع الگوریتم وارد شده، از الگوریتم های مناسب برای پیدا کردن بهترین جواب استفاده میکند.

از آنجایی که فضای حالت بازی ما ۲۵ حالت دارد، باید یک عمقی تعیین کنیم چرا که درخت با ۲۵ حالت در فضایش، بسیار بزرگ میشود و قابل بررسی نیست.

همان طور که میدانیم الگوریتم هرس شده ی minimax بسیار بهتر از الگوریتم minimax کار میکند برای اثبات هم یک متغیر به اسم num_repeat_algorithm تعریف شده که هرزمان از یک الگوریتم استفاده کنیم، یکی به این متغیر اضافه میشود. با نزدیک تر شدن به جواب و کاهش عمق و بالا امدن از درخت (چون بازگشتی کار میکنیم از انتهای درخت شروع میشود)، تعدادی که این متغیر نشان میدهد کمتر میشود چرا که تعداد نودکمتری بررسی میشود.

با مقایسه ی مقدار این متغیر در الگوریتم هرس شده minimax با خود minimax، متوجه سریع تر بودن و کارامد بودن آن میشویم.

اگر از نظر زمان هم بخاهیم مقایسه کنیم، زمانی که الگوریتم هرس شده طول میکشد تا جواب را پیدا کند بسیار کمتر است.

در این برنامه، از عمق ۵ برای الگوریتم minimax و از عمق ۸ برای الگوریتم هرس شده minimax استفاده شده است و این اعداد به صورت تجربی بدست امدند (عمق های بیشتر بسیار زمان بیشتری مصرف میکردند تا جواب را حساب کنند)

```python
def ai_move(board, player, algorithm):  # max player
    global num_repeat_algorithm
    num_repeat_algorithm = 0

    if algorithm == MINIMAX_ALGORITHM:
        move, _ = minimax(board, player, 5)
        algorithm_str = "MINIMAX"

    elif algorithm == MINIMAX_AB_ALGORITHM:
        move, _ = minimax_ab(board, player, 8)
        algorithm_str = "alpha&beta MINIMAX"

    if move is None:
        print(colored(
            f'** in {algorithm_str} algorithm, move was none so a random number selected**'))
        move = randint(0, 24)

    board[move] = player
    print(colored(
        f"\nnumber of Calls to {algorithm_str} function is: {num_repeat_algorithm}", MESSAGES_COLOR1))
```

تابع بازی

```
def game(board, player1, player2, game_type, game_algorithm=""):
    global is_human_first_player
    turn = player1
    is_quit_game = False
    while not is_game_over(board):
        print(colored("\nThis is the current board state:", MESSAGES_COLOR1))
        print_board(board)
        print(colored(f"\n\tIt is \'{turn}\' player's move:", MESSAGES_COLOR2))
        if (game_type == HUMAN_VS_AI_GAME):
            if (turn == player1 and is_human_first_player):
                if human_move(turn) == "q":
                    is_quit_game = True
                    break
            elif (turn == player2):
                ai_move(board, turn, game_algorithm)
                is_human_first_player = True

        elif (game_type == AI_VS_AI_GAME):
            ai_move(board, turn, game_algorithm)

        if turn == player1:
            turn = player2
        else:
            turn = player1

    if is_quit_game:
        return

    winner = check_win(board)
    if winner in (G_player, R_player):
        print_board(board)
        winner_congrats(winner)
    elif winner == "draw":
        print("It's a draw!")
    return
```

در این تابع بدنه ی اصلی بازی اجرا میشود.

تا زمانی که بازی تمام نشده (گیم اور نشده) در حلقه هر بار نوبت یک بازیکن میشود.

بسته به اینکه بازی از چه نوعی است، حرکت مناسب انجام میشود. اگر بازی از جنس انسان در برابر کامپیوتر باشد، بررسی میشود که آیا حرکت اول برای انسان است یا خیر.

اگر هم بازی از جنس دوم باشد که هربار تابع ai_move فراخوانی میشود و بسته به الگوریتم انتخاب شده، بهترین حرکت برای کامپیوترها انتخاب میشود.

در انتها هم نوبت بازی تغییر میکند و اگر نوبت بازیکن اول بوده، نوبت به دومی داده میشود و به همین صورت برای دیگری.

اگر بازی تمام شود از این حلقه خارج میشویم. اگر برنده داشته باشیم پیامی چاپ میشود که بازیکن برنده کدام است.